目　录

第一部分　入门 —— Simulink 辅助设计基础

第二部分　进阶 —— Simulink 动态系统仿真

第三部分 精通——Simulink 高级仿真技术

第四部分　实例分析——利用 Simulink 进行系统设计与分析

第一部分

入门

Simulink 辅助设计基础

MATLAB程序用丛书

第一部分主要介绍动态系统仿真技术的基础知识。如果读者对动态系统仿真技术与 MATLAB 的相关知识都比较熟悉，则可以直接学习第二部分的内容。不过建议读者最好能够浏览一下第一部分的内容，或许你会有新的发现。本部分包括如下内容：

➤**绪论：主要介绍系统仿真的基本概念。**

➤**Simulink 使用基础：主要介绍 Simulink 使用的 MATLAB 基础。**

➤**动态系统模型及其 Simulink 表示：主要介绍动态系统的数学描述及其在 Simulink 中的表示方法。**

第1章　绪　论

内容概要

· 动态系统仿真的基本概念与基本知识
· 使用 Simulink 进行动态系统的仿真
· 本书的组织结构

1.1　动态系统的计算机仿真

1.1.1　系统与模型

为了能全面、正确地理解系统仿真，需要对系统仿真所研究的对象进行概要的了解。这里对与系统仿真相关的知识——系统与系统模型进行简单的介绍。

1. 系统

系统是指具有某些特定功能，相互联系、相互作用的元素的集合。这里的系统是指广义上的系统，泛指自然界的一切现象与过程。它具有两个基本特征：整体性和相关性。整体性是指系统作为一个整体存在而表现出某项特定的功能，它是不可分割的。相关性是指系统的各个部分、元素之间是相互联系的，存在物质、能量与信息的交换。在自然界与人类社会中存在着多种系统，既包含工程系统如控制系统、通信系统等，也包含非工程系统如股市系统、交通系统、生物系统等。对于任何系统的研究都必须从如下三个方面考虑：

(1) 实体：组成系统的元素、对象。

(2) 属性：实体的特征。

(3) 活动：系统由一个状态到另一个状态的变化过程。

组成系统的实体之间相互作用而引起的实体属性的变化，通常用状态变量来描述。研究系统主要研究系统的动态变化。除了研究系统的实体属性活动外，还需要研究影响系统活动的外部条件，这些外部条件称之为环境。自然界的实物之间是普遍联系的，系统是在外界环境的不断变化中产生活动的，因此研究系统所处的环境是十分必要的。但是应该注意到，系统与环境的边界是不确定的，对于同一个系统可能因为研究目的的不同而不同。系统仿真是研究系统的一种重要手段，而系统模型则是仿真所要研究的直接对象。

2. 系统模型

系统模型是对实际系统的一种抽象，是对系统本质(或是系统的某种特性)的一种描述。模型可视为对真实世界中物体或过程的信息进行形式化的结果。模型具有与系统相似的特性，可以以各种形式给出我们所感兴趣的信息。好的模型能够更深刻地反映实际系统的主

要特征和运动规律，它是对实际系统更高层次上的抽象，本身就是对实体认识的结果。从这个意义上来说，模型优于实体。

模型可以分为实体模型和数学模型。实体模型又称为物理效应模型，是根据系统之间的相似性而建立起来的物理模型。实体模型最常见的是比例模型，如风洞吹风实验常用的翼型模型或建筑模型。数学模型包括原始系统数学模型和仿真系统数学模型。原始系统数学模型是对系统的原始数学描述。仿真系统数学模型是一种适合在计算机上演算的模型，主要是指根据计算机的运算特点、仿真方式、计算方法、精度要求将原始系统数学模型转换为计算机程序。

数学模型可以分为许多类型。按照状态变化可分为动态模型和静态模型。用以描述系统状态变化过程的数学模型称为动态模型。而静态模型仅仅反映系统在平衡状态下系统特征值间的关系，这种关系常用代数方程来描述。按照输入和输出的关系可分为确定性模型和随机性模型。若一个系统的输出完全可以用它的输入来表示，则称之为确定性系统。若系统的输出是随机的，即对于给定的输入存在多种可能的输出，则该系统是随机系统。作为主要研究对象的动态系统又可分为连续系统和离散系统。连续系统的动态模型常用微分方程、状态方程或传递函数来描述。研究这些系统的性质实际上就是求解微分方程。离散系统是指系统的操作和状态变化仅在离散时刻产生的系统，如交通系统、电话系统、通信网络系统等等，常常用各种概率模型来描述。连续系统模型还可分为集中参数的和分布参数的，线性的和非线性的，时变的和时不变的，时域的和频域的，连续时间的和离散时间的等等。表 1.1 列出了各种类型的数学模型及其数学描述。

表 1.1 数学模型分类

模型类型	静态系统模型	动态系统模型			
		连续系统模型			离散系统模型
		集中参数	分布参数	离散时间	
数学描述	代数方程	微分方程 状态方程 传递函数	偏微分方程	差分方程 离散状态方程	概率分布 排队论

1.1.2 计算机仿真

1. 仿真的概念

仿真是以相似性原理、控制论、信息技术及相关领域的有关知识为基础，以计算机和各种专用物理设备为工具，借助系统模型对真实系统进行试验研究的一门综合性技术。它利用物理或数学方法来建立模型，类比模拟现实过程或者建立假想系统，以寻求过程的规律，研究系统的动态特性，从而达到认识和改造实际系统的目的。

系统仿真涉及相似论、控制论、计算机科学、系统工程理论、数值计算、概率论、数理统计、时间序列分析等多种学科。

相似性原理是仿真主要的理论依据。所谓相似，是指各类事务或对象间存在的某些共性。相似性是客观世界的一种普遍现象，它反映了客观世界不同事物之间存在着某些共同的规律。采用相似性技术建立实际系统的相似模型就是仿真的本质过程。相似性原理的基本内容包括相似性定义、相似定理和相似方法。因为系统本身具有内部结构和外部行为，所以系统相似性分为两个层次：结构层次和行为层次。具有相同的内部结构必然具有行为

等价的特性，但是行为上的等价并不能说明两个系统具有同构关系。一般而言，对现实世界的模拟都归结为或体现在外部行为的等价。

2. 仿真分类

按照实现方式的不同可以将系统仿真分为如下几类：

(1) 实物仿真：又称物理仿真。它是指研制某些实体模型，使之能够重现原系统的各种状态。早期的仿真大多属于这一类。它的优点是直观形象，至今仍然广泛应用。但是为系统构造一套物理模型，将是一件非常复杂的事情，投资巨大，周期长，且很难改变参数，灵活性差。至于社会系统、经济系统、生物系统则根本就无法用实物作实验。实物仿真系统有各种风洞、水洞以及著名的生态圈Ⅰ等。例如为了研究飞机翼型，要建立翼型的比例模型，更重要的是要在地面建立对空中气流环境的模拟，投资巨大，周期长，且灵活性差，但是为了研制、验证一种新的翼型，这又往往是必不可少的。

(2) 数学仿真：数学仿真就是用数学语言去表述一个系统，并编制程序在计算机上对实际系统进行研究的过程。这种数学表述就是数学模型。数学仿真把研究对象的结构特征或者输入输出关系抽象为一种数学描述(微分方程、状态方程，可分为解析模型、统计模型)来研究，具有很大的灵活性，它可以方便地改变系统结构、参数；而且速度快，可以在很短的时间内完成实际系统很长时间的动态演变过程；精确度高，可以根据需要改变仿真的精度；重复性好，可以很容易地再现仿真过程。然而数学仿真也有其局限性。对某些复杂系统可能很难用数学模型来表达，或者难以建立其精确模型，或者数学模型过于复杂而目前无法求解，或者计算量太大而无法利用现有的计算资源进行仿真。

(3) 半实物仿真：又称数学物理仿真或者混合仿真。为了提高仿真的可信度或者针对一些难以建模的实体，在系统研究中往往把数学模型、物理模型和实体结合起来组成一个复杂的仿真系统，这种在仿真环节中存在实体的仿真称为半实物仿真或者半物理仿真。这样的仿真系统有飞机半实物仿真、射频制导导弹半实物仿真等，并且许多模拟器也属于半实物仿真。

实际上在工程实践中，以上各种仿真往往用于工程中的不同阶段。在工程设计分析阶段采用数学仿真，易于更改设计，具有灵活性和经济性。在部件子系统研制阶段，采用半实物仿真以提高仿真可信度和测试部件或子系统的功能。在最后定型阶段为了验证全系统的功能特性，则需要进行全物理仿真。

按照仿真系统与实际系统时间尺度上的关系，又可将其分为如下几类：

(1) 实时仿真：仿真时钟与系统实际时钟完全一致。许多仿真应用需要满足实时性，这时往往需要实时操作系统或者专用实时仿真硬件的支持。

(2) 欠实时仿真：仿真时钟比实际时钟慢。当对仿真的实时性没有严格的要求时，仿真时钟比实际时钟慢，不影响仿真的目的，采取欠实时仿真则可节约很多资金。

(3) 超实时仿真：仿真时钟比实际时钟快。当实际系统周期太长时，若采用实际时钟就变得毫无意义，这时就要进行超实时仿真。对于大的、复杂的系统进行超实时仿真对计算机的计算速度要求是非常高的，如天气预报系统就需要超级计算机的支持。

3. 计算机仿真

计算机仿真是在研究系统过程中根据相似原理，利用计算机来逼真模拟研究对象。研究对象可以是实际的系统，也可以是设想中的系统。在没有计算机以前，仿真都是利用实

物或者它的物理模型来进行研究的，即物理仿真。物理仿真的优点是直接、形象、可信，缺点是模型受限、易破坏、难以重用。而计算机仿真是将研究对象进行数学描述、建模编程，且在计算机中运行实现。它不怕破坏、易修改、可重用。计算机仿真可以用于研制产品或设计系统的全过程中，包括方案论证、技术指标确定、设计分析、生产制造、试验测试、维护训练、故障处理等各个阶段。

计算机作为一种最重要的仿真工具，已经推出了模拟机、模拟数字机、数字通用机、仿真专用机等各种机型并应用在不同的仿真领域。除了计算机这种主要的仿真工具外还有两类专用仿真器：一类是专用物理仿真器，如在飞行仿真中得到广泛应用的转台，各种风洞、水洞等；另一类是用于培训目的的各种训练仿真器，如培训原子能电站、大型自动化工厂操作人员和训练飞行员、宇航员的培训仿真器、仿真工作台和仿真机舱等。训练仿真器实际上是一种包括计算机、物理模型、实物在内的复杂仿真系统。

1.1.3 仿真的作用

仿真技术具有很高的科学研究价值和巨大的经济效益。由于仿真技术的特殊功效，特别是安全性和经济性，使得仿真技术得到广泛的应用。首先由于仿真技术在应用上的安全性，使得航空、航天、核电站等成为仿真技术最早的和最主要的应用领域。特别是在军事领域，新型的武器系统、大型的航空航天飞行器在其设计、定型过程中，都要依靠仿真试验进行修改和完善：导弹、火箭的设计研制，空战、电子战、攻防对抗等演练也都离不开仿真技术。其次从仿真的经济性考虑，由于仿真往往是在计算机上模拟现实系统过程，并可多次重复运行，使得其经济性十分突出。据美国对“爱国者”等三个型号导弹的定型试验统计，采用仿真试验可减少实弹发射试验次数约 43%，节省费用达数亿美元。我国某种型号导弹在设计和定型过程中，通过仿真试验就缩短研制时间近两年，少进行 20 多次实弹射击，节省费用数千万元。如果不进行仿真试验，导弹改型一次，就要重新进行多次实弹发射，型号定型往往需要进行数十次甚至上百次发射试验。采用模拟器培训工作人员，经济效益和社会效益也十分明显。另外，从环境保护的角度考虑，仿真技术也极具价值。例如，现代核试验多是在计算机上仿真进行，固然是由于计算机技术的发展使其得以在计算机上模拟，但政治因素和环境因素才是进行核试验仿真的主要原因。

仿真技术在许多复杂工程系统的分析和设计研究中越来越成为不可缺少的工具。系统的复杂性主要体现在复杂的环境、复杂的对象和复杂的任务上。然而只要能够正确地建立系统的模型，就能够对该系统进行充分的分析研究。另外，仿真系统一旦建立就可重复利用，特别是对计算机仿真系统的修改非常方便。经过不断的仿真修正，逐渐深化对系统的认识，以采用相应的控制和决策，使系统处于科学的控制和管理之下。

归纳起来，仿真技术的主要用途有如下几点：

(1) 优化系统设计。在实际系统建立以前，通过改变仿真模型结构和调整系统参数来优化系统设计。如控制系统、数字信号处理系统的设计经常要靠仿真来优化系统性能。

(2) 系统故障再现，发现故障原因。实际系统故障的再现必然会带来某种危害性，这样做是不安全的和不经济的，利用仿真来再现系统故障则是安全的和经济的。

(3) 验证系统设计的正确性。

(4) 对系统或其子系统进行性能评价和分析。多为物理仿真，如飞机的疲劳试验。

(5) 训练系统操作员。常见于各种模拟器，如飞行模拟器、坦克模拟器等。

(6) 为管理决策和技术决策提供支持。

1.1.4　仿真算法和仿真软件

1. 仿真算法

在建立系统的数学模型后，需要将其转变成能够在计算机上运行的仿真模型。由于计算机只能进行离散的数值计算，因而必须推导出连续系统的递推数学公式，如解微分方程的龙格库塔算法。这实际上属于数值计算的内容，其发展已经相当完善了。其实这就是计算机仿真算法的设计，即把数学模型转化为能在计算机上运行的仿真模型。

通常这些仿真算法并不需要仿真人员去编制，因为这些仿真算法往往已经内嵌于各种面向仿真用途的专用软件中了。但是对这些算法的了解无疑有助于用户更好地完成仿真任务。一般来说，系统仿真算法有如下几类：

(1) 集中参数系统仿真算法。

(2) 分布参数系统仿真算法。

(3) 离散时间系统仿真算法。

2. 仿真软件

仿真软件是一类面向仿真用途的专用软件，它可能是面向通用的仿真，也可能是面向某个领域的仿真。它的功能可以概括为以下几点：

(1) 为仿真提供算法支持。

(2) 模型描述，用来建立计算机仿真模型。

(3) 仿真实验的执行和控制。

(4) 仿真数据的显示、记录和分析。

(5) 对模型、实验数据、文档资料和其它仿真信息的存储、检索和管理(即用于仿真数据信息管理的数据库系统)。

根据软件功能，仿真软件可分为以下三个层次：

(1) 仿真程序库：由一组完成特定功能的程序组成的集合，专门面向某一问题或某一领域。它可能是用通用的语言(C++、FORTRAN 等)开发的程序软件包，也可能是依附于某种集成仿真环境的函数库或模块库。很多这样的软件包都是免费的(可以在网上下载)，包括 C++SIM，Mathtools(for MATLAB、C、C++、FORTRAN)，SolutionBase(for Delphi)等。

(2) 仿真语言：仿真语言多属于面向专门问题的高级语言，它是针对仿真问题，在高级语言的基础上研制的。它不要求用户掌握复杂繁琐的高级语言，只需用户按照要求书写方程代码，而无需考虑数学模型到仿真模型的转换，这种代码往往更接近于系统本身的数学模型。最终，由机器自动完成由仿真语言到通用高级语言或汇编语言的转换。这样的语言有 ACSL，Simscript，Easy5，Adsim 等。

(3) 集成仿真环境：它是一组用于仿真的软件工具的集合，包括设计、分析、编制系统模型，编写仿真程序，创建仿真模型，运行、控制、观察仿真实验，记录仿真数据，分析仿真结果，校验仿真模型等。它涉及到许多功能软件，如建模软件、仿真执行软件、结果分析软件等。各功能软件存在着信息联系。为了提高效率，必须将它们集成起来，加上方便的操作界面、环境，就形成了集成仿真环境。MathWorks 公司的 Simulink 就是这样的软

件系统。此外还有 Matrix(已经被 MathWorks 并购)、Mideva、Scilab 等集成开发环境。

用户可以链接http://www.idsia.ch/~andrea/simtools.html以查阅各种仿真软件的简介，对系统仿真软件进行进一步的了解。

1.1.5 计算机仿真的一般过程

计算机仿真的一般过程可以表述如下：

(1) 描述仿真问题，明确仿真目的。

(2) 项目计划、方案设计与系统定义。根据仿真目的确定相应的仿真结构(实时仿真还是非实时仿真，纯数学仿真还是半物理仿真等)，规定相应仿真系统的边界条件与约束条件。

(3) 数学建模：根据系统的先验知识、实验数据及其机理研究，按照物理原理或者采取系统辨识的方法，确定模型的类型、结构及参数。注意要确保模型的有效性和经济性。

(4) 仿真建模：根据数学模型的形式、计算机类型、采用的高级语言或其它仿真工具，将数学模型转换成能在计算机上运行的程序或其他模型，也即获得系统的仿真模型。

(5) 试验：设定实验环境/条件和记录数据，进行实验，并记录数据。

(6) 仿真结果分析：根据实验要求和仿真目的对实验结果进行分析处理(整理及文档化)。根据分析结果修正数学模型、仿真模型或仿真程序或者修正/改变原型系统，以进行新的实验。模型是否能够正确地表示实际系统，并不是一次完成的，而是需要比较模型和实际系统的差异，不断地修正和验证而完成的。

图 1.1 所示为计算机仿真过程流程图。

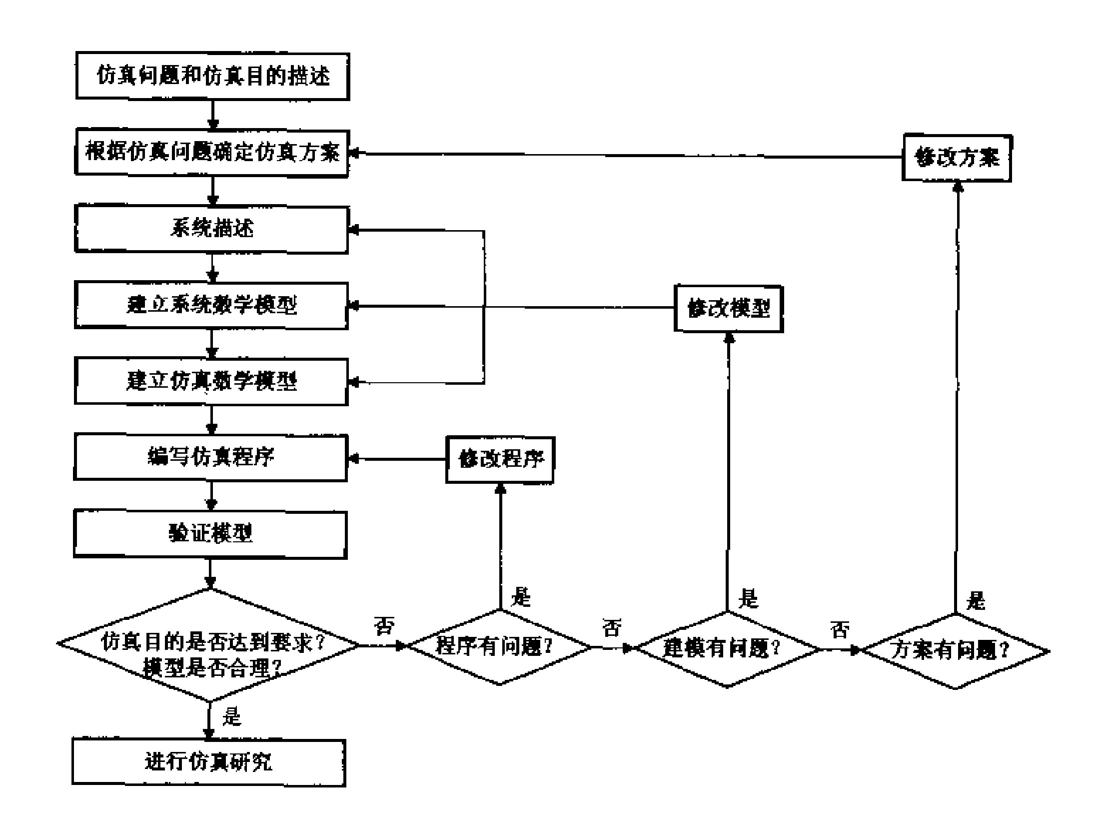


图 1.1　计算机仿真流程图

1.2 动态系统的 Simulink 仿真

1.2.1 Simulink 简介

Simulink 是一个用来对动态系统进行建模、仿真和分析的软件包。使用 Simulink 来建模、分析和仿真各种动态系统(包括连续系统、离散系统和混合系统)，将是一件非常轻松的事情。它提供了一种图形化的交互环境，只需用鼠标拖动的方法便能迅速地建立起系统框图模型，甚至不需要编写一行代码。它和 MATLAB 的无缝结合使得用户可以利用 MATLAB 丰富的资源，建立仿真模型，监控仿真过程，分析仿真结果。另外，Simulink 在系统仿真领域中已经得到广泛的承认和应用，许多专用的仿真系统都支持 Simulink 模型，这非常有利于代码的重用和移植。使用 Simulink 可以方便地进行控制系统、DSP 系统、通信系统以及其它系统的仿真分析和原型设计。

利用 Simulink 进行系统的建模仿真，其最大的优点是易学、易用，并能依托 MATLAB 提供的丰富的仿真资源。这里对 Simulink 的强大功能进行简单的介绍。

1. 交互式、图形化的建模环境

Simulink 提供了丰富的模块库以帮助用户快速地建立动态系统模型。建模时只需使用鼠标拖放不同模块库中的系统模块并将它们连接起来。另外，还可以把若干功能块组合成子系统，建立起分层的多级模型，Simulink 4.1 版提供的模型浏览器(model browser)可以使用户方便地浏览整个模型的结构和细节。Simulink 这种图形化、交互式的建模过程非常直观，且容易掌握。

2. 交互式的仿真环境

Simulink 框图提供了交互性很强的仿真环境，既可以通过下拉菜单执行仿真，也可以通过命令行进行仿真。菜单方式对于交互工作非常方便，而命令行方式对于运行一大类仿真如蒙特卡罗仿真非常有用。有了 Simulink，用户在仿真的同时，可采用交互或批处理的方式，方便地更换参数来进行"What-if"式的分析仿真。仿真过程中各种状态参数可以在仿真运行的同时通过示波器或者利用 ActiveX 技术的图形窗口显示。

3. 专用模块库(Blocksets)

作为 Simulink 建模系统的补充，MathWorks 公司还开发了专用功能块程序包，如 DSP Blockset 和 Communication Blockset 等。通过使用这些程序包，用户可以迅速地对系统进行建模、仿真与分析。更重要的是用户还可以对系统模型进行代码生成，并将生成的代码下载到不同的目标机上。可以说，MathWorks 为用户从算法设计、建模仿真，一直到系统实现提供了完整的解决方案。而且，为了方便用户系统的实施，MathWorks 公司还开发了实施软件包，如 TI 和 Motorola 开发工具包，以方便用户进行目标系统的开发。表 1.2 列出了 Simulink 的一些软件工具包。

表 1.2 Simulink 的部分软件工具包

DSP Blockset	数字信号处理工具包
Fixed-Point Blockset	定点运算控制系统仿真工具包
Power System Blockset	电力电动系统工具包
Dials & Gauges Blockset	交互图形和控制面板设计工具包
Communications Blockset	通讯系统工具包
CDMA Reference Blockset CDMA	CDMA 通讯系统设计和分析工具包
Nonlinear Control Design Blockset	非线性控制设计工具箱
Motorola DSP Developer's Kit	Motorola DSP 开发工具箱
TI DSP Developer's Kit	TI DSP 开发工具箱

4. 提供了仿真库的扩充和定制机制

Simulink 的开放式结构允许用户扩展仿真环境的功能：采用 MATLAB、FORTRAN 和 C 代码生成自定义模块库，并拥有自己的图标和界面。因此用户可以将使用 FORTRAN 或 C 编写的代码链接进来，或者购买使用第三方开发提供的模块库进行更高级的系统设计、仿真与分析。

5. 与 MATLAB 工具箱的集成

由于 Simulink 可以直接利用 MATLAB 的诸多资源与功能，因而用户可以直接在 Simulink 下完成诸如数据分析、过程自动化、优化参数等工作。工具箱提供的高级的设计和分析能力可以融入仿真过程。

简而言之，Simulink 具有以下特点：

(1) 基于矩阵的数值计算。

(2) 高级编程语言。

(3) 图形与可视化。

(4) 工具箱提供面向具体应用领域的功能。

(5) 丰富的数据 I/O 工具。

(6) 提供与其它高级语言的接口。

(7) 支持多平台(PC / Macintosh / UNIX)。

(8) 开放与可扩展的体系结构。

1.2.2 Simulink 的应用领域

至此，读者应该对动态系统的模型建立、系统仿真与分析有了一个比较感性的认识；同时对 Simulink 的强大功能也会有一定的了解。那么使用 Simulink 到底可以对什么样的动态系统进行仿真分析与辅助设计呢？其实，任何使用数学方式进行描述的动态系统都可以使用 Simulink 进行建模、仿真与分析。

由于 Simulink 具有强大的功能与友好的用户界面，因此它已经被广泛地应用到诸多领域之中，如：

(1) 通讯与卫星系统。

(2) 航空航天系统。
(3) 生物系统。
(4) 船舶系统。
(5) 汽车系统。
(6) 金融系统。

此外，Simulink 在生态系统、社会和经济等领域也都有所应用。在科学技术飞速发展的 21 世纪，Simulink 的应用领域也将会更加广泛。图 1.2 所示为 Simulink 在一些领域中的典型应用。

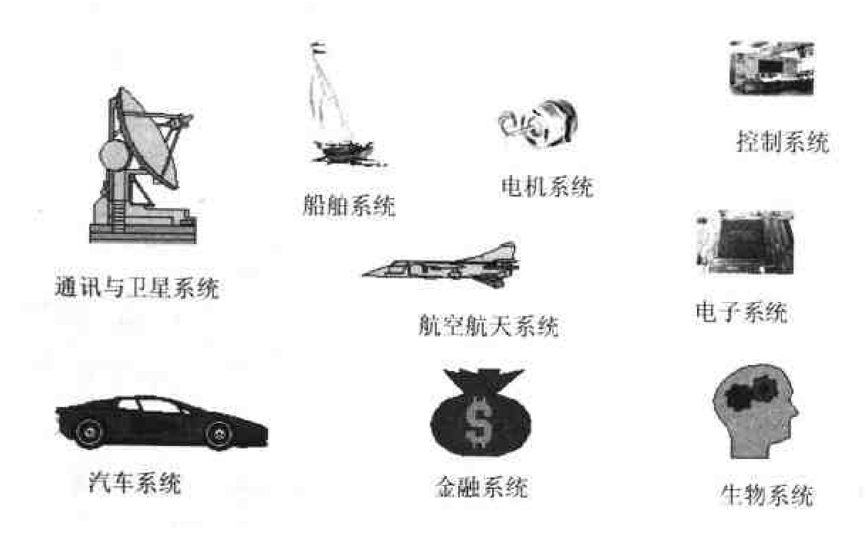


图 1.2 Simulink 的应用领域示意图

1.2.3 Simulink 在 MATLAB 家族中的位置

MATLAB 是一个包含数值计算、高级图形与可视化、高级编程语言的集成化科学计算环境。MATLAB Toolbox 提供了面向专业的函数库，扩展了 MATLAB 的能力。MATLAB Compiler 自动将 MATLAB 中的 M 文件转换成 C 和 C++代码，用于独立应用开发。Simulink 是一个交互式动态系统建模、仿真和分析工具。Simulink Blockset 提供了丰富的专业模块库，广泛地用于控制、DSP、通讯等系统仿真领域。Stateflow 是一种利用有限状态机理论建模和仿真事件驱动系统的可视化设计工具，适合用于描述复杂的开关控制逻辑、状态转移图以及流程图等。Real-Time Workshop 能够从 Simulink 模型中生成可定制的代码及独立的可执行程序。Stateflow coder 能够自动生成状态图的代码，并且能够自动地结合到 RTW 生成的代码中。图 1.3 所示为 Simulink 与 MATLAB 的层次结构示意图。

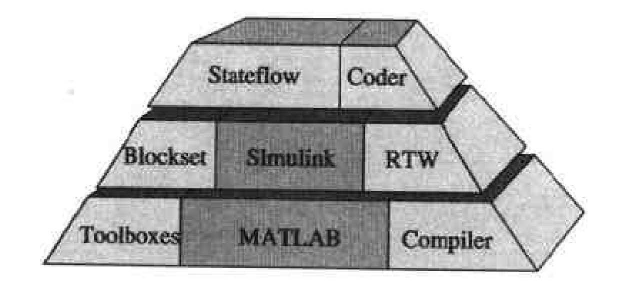


图 1.3 Simulink 与 MATLAB 之间的层次关系示意图

1.3 本书的组织结构

在对 Simulink 做进一步的介绍之前，首先对本书的组织结构进行简单的介绍。不同的用户可以根据自己的专业背景、对系统仿真认识的程度以及对使用 Simulink 进行动态系统仿真掌握的程度来制定本书的使用方法。

在本书的编排之中，我们竭力使每一章的内容在一定程度上都能够自成体系，以方便不同用户的需要。在使用 Simulink 进行动态系统模型的建立、仿真与分析时，用户可以随时查阅本书的相应内容，以满足特定的需要。本书既可作为学习 Simulink 的教材，也可作为一本使用手册供读者查阅。

图 1.4 所示为本书的组织结构说明。

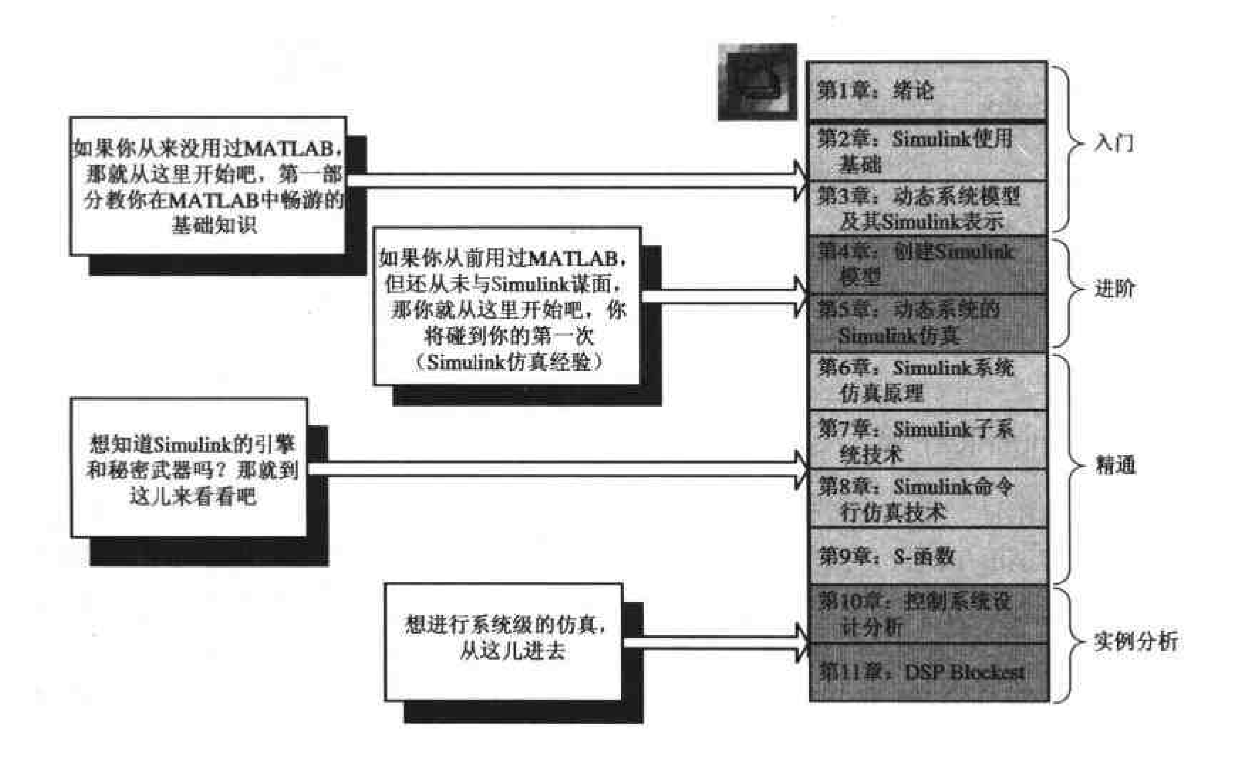


图 1.4 本书的组织结构说明

第 2 章 Simulink 使用基础

内容概要

- MATLAB 的基本数据类型：向量与矩阵
- MATLAB 中的基本操作与运算
- 多项式操作与运算
- MATLAB 的图形绘制
- MATLAB 函数与 M 文件

Simulink 是 MATLAB 的一个交互式动态系统建模、仿真和分析工具，它具有高性能的科学计算能力，被广泛应用于线性、非线性、离散、连续及多变量系统的仿真和分析。由于它的运行需要 MATLAB 环境的支持，因此在介绍 Simulink 建模、仿真及分析之前，有必要先了解一些与之相关的 MATLAB 的基础知识。

2.1 MATLAB 的计算单元：向量与矩阵

MATLAB 作为一个高性能的科学计算平台，主要面向高级科学计算。MATLAB 的基本计算单元是矩阵与向量，向量为矩阵的特例。一般而言，二维矩阵为由行、列元素构成的矩阵表示；对于 m 行、n 列的矩阵，其大小为 $m\times n$。在 MATLAB 中表示矩阵与向量的方法很直观，下面举例说明。

例如，矩阵 $\boldsymbol{A}=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$，行向量 $\boldsymbol{B}=\begin{bmatrix}1 & 2 & 3\end{bmatrix}$，列向量 $\boldsymbol{C}=\begin{bmatrix}4\\5\\6\end{bmatrix}$，在 MATLAB 中可以分别表示为

```
>>A=[1 2 3; 4 5 6]
>>B=[1 2 3]
>>C=[4; 5; 6]
```

注意：(1) MATLAB 中所有的矩阵与向量均包含在中括号[]之中。如果矩阵的大小为 1 × 1，则它表示一个标量，如

```
>>a=3                    %a 表示一个数
```

(2) 矩阵与向量中的元素可以为复数，在 MATLAB 中内置虚数单元为 i、j；虚数的表达很直观，如 3 + 4*i 或者 3 + 4*j 。

技巧：(1) MATLAB 中对矩阵或向量元素的引用方式与通常矩阵的引用方式一致，如 A(2 ,3)表示矩阵 A 的第 2 行第 3 列的元素。如若对 A 的第 2 行第 3 列的元素重新赋值，只需键入如下命令：

```
>>A(2,3)=8;
```

则矩阵 A 变为

```
A =
    1    2    3
    4    5    8
```

(2) MATLAB 中分号(;)的作用有两点：一是作为矩阵或向量的分行符，二是作为矩阵或向量的输出开关控制符。即如果输入矩阵或向量后键入分号，则矩阵与向量不在 MATLAB 命令窗口中显示，否则将在命令窗口中显示。如输入矩阵

```
>>A=[1 2 3; 4 5 6]            % 按下 Enter 键，则在 MATLAB 命令窗口中显示
>>A =
    1    2    3
    4    5    6
```

除了对二维矩阵的支持之外，MATLAB 还支持包括三维矩阵在内的多维矩阵，对于多维矩阵的表示也很直观，如由二维矩阵 $\boldsymbol{D}=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$ 与 $\boldsymbol{E}=\begin{bmatrix}0 & 2 & 5\\3 & 7 & 8\end{bmatrix}$ 构成的三维矩阵 $\boldsymbol{F}$ 可以表示为

```
>> F(:,:, 1)=D;
>> F(:,:, 2)=E;
```

即

```
>> F(:,:, 1)=[1 2 3; 4 5 6];
>> F):,:, 2)=[0 2 5; 3 7 8];
```

对于维数大于 3 的多维矩阵的建立和表达与此相似，对多维矩阵的引用和赋值与二维矩阵类似，这里不再赘述。

(3) 冒号操作符(:)的应用。冒号操作符在建立矩阵的索引与引用时非常方便且直接。如上述对多维矩阵 $\boldsymbol{F}$ 的建立中，冒号操作符表示对矩阵 $\boldsymbol{F}$ 第一维与第二维所有元素按照其顺序进行引用，从而对 $\boldsymbol{F}$ 进行快速赋值，无需一一赋值。如

```
>>B=2:5                   %对向量进行赋值
>>B=
    2  3  4  5
>>B（1:3）=2              %向量 B 从第 1 个到第 3 个元素全部赋值为 2
>>B=
    2  2  2  5
>> C=6: -2:0              %将向量 C 进行递减赋值，初始值为 6，终止值为 0，步长为－2
>>C=
    6  4  2  0
```

冒号操作符的使用很灵活，如图 2.1 所示。

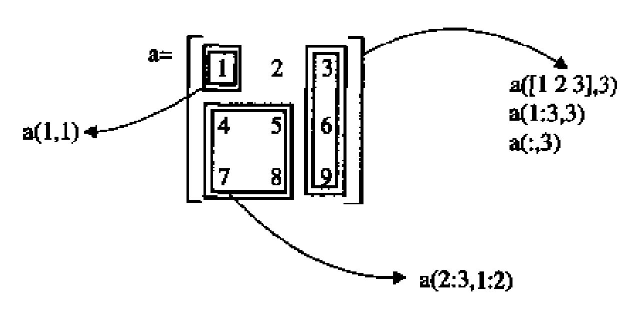


图 2.1　冒号操作符使用示意图

从图 2.1 中可以看出，使用冒号操作符对矩阵元素进行引用非常灵活和方便，它可以有效地对矩阵的指定元素或指定区域进行各种操作与控制。这使得 MATLAB 对矩阵的操作方式非常符合习惯的用法，易于理解与应用。此外，MATLAB 还支持多种不同类型的数据，它们的建立和基本引用与本节介绍的基本相同，这里不再赘述。有兴趣的读者可以参考 MATLAB 的联机帮助。

2.2　MATLAB 计算单元的基本操作

2.1 节介绍了 MATLAB 的基本计算单元，即矩阵与向量的建立与引用方法。本节将简单介绍在 MATLAB 环境下矩阵与向量的操作与运算。

1. 矩阵加法与减法

如果矩阵 $\boldsymbol{A}$ 与矩阵 $\boldsymbol{B}$ 具有相同的维数，则可以定义矩阵的加法与减法，其结果为矩阵相应元素作运算所构成的矩阵。矩阵加法与减法在 MATLAB 中的表达方式为

```
>> C=A+B;                    %C 为矩阵 A 与 B 之和
>> D=A-B;                    %D 为矩阵 A 与 B 之差
```

【例 2.1】　若 $\boldsymbol{A}=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$，$\boldsymbol{B}=\begin{bmatrix}0 & 2 & 1\\2 & 5 & 3\end{bmatrix}$，则

```
>>C=
    1    4    4
    6   10    9
>>D=
    1    0    2
    2    0    3
```

矩阵与标量的加法与减法是指标量本身与矩阵所有元素进行相应运算，如若 $b=1$，$\boldsymbol{E}=\boldsymbol{A}+b$，则

```
>>E=
    2    3    4
    5    6    7
```

2. 矩阵的乘法与除法

如果矩阵A的列数等于矩阵B的行数，则矩阵A、B可以相乘。其结果C=AB在MATLAB中可表示为

```
>>C=A*B;              %A、B相乘，若A、B不满足矩阵乘法法则，MATLAB会给出出错信息
```

【例 2.2】 若$A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}$，$B=\begin{bmatrix}1 & 2\\ 0 & 3\\ 2 & 1\end{bmatrix}$，则

```
>>C=
     7   11
    16   29
```

如果矩阵A为方阵，A的p次方可以用A^p表示。如果p是一个正整数，那么这个幂可以由矩阵的连续相乘定义。当p=0时，其结果为与A相同的矩阵；当p<0时，只有在A的逆存在时才可定义A^p，其意义为inv(A)^($-p$)。

在MATLAB中，矩阵除法有两种形式，即左除(\)和右除(/)。如果A是一个非奇异方阵，那么

```
>>A\B                 % 表示A的逆与B的左乘，即inv(A)*B
>>B/A                 % 表示A的逆与B的右乘，即B* inv(A)
```

矩阵的左除和右除运算还可以用来求解矩阵方程AX=B的解：

```
>>X=A\B
```

如果A是一个方阵，X就是方程的解；如果A是一个行数大于列数的矩阵，X就是方程的最小二乘解。

3. 矩阵的转置

转置是一种重要的矩阵运算，在MATLAB中由撇号表示：

```
>> B=A'               %B为A的转置
```

如果$A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}$，则$B=\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}$。如果$A$中含有复数元素，则$A$的转置矩阵中的元素为原来元素的共轭。

4. 对矩阵元素的操作与运算

在上述各种常用运算中，所有的操作都是针对矩阵所有元素或一部分元素的操作。其实还可以对矩阵元素进行单独的操作运算。对于加法和减法，对矩阵元素的操作与对矩阵的操作是一致的。其它运算对于所有矩阵元素的操作需要在操作符前加点，如【例 2.3】所示。

【例 2.3】 若$A=\begin{bmatrix}1 & 2\\ -1 & 5\end{bmatrix}$，$B=\begin{bmatrix}7 & 2\\ 1 & 0\end{bmatrix}$，$C=\begin{bmatrix}1+2i & 5-2i\\ 3+i & 1+3i\end{bmatrix}$，则

```
>>A.*B=               % 矩阵对应元素相乘
```

```
      7      4
      -1     0
  >>B./A=                % 矩阵对应元素相除
      7      1
      -1     0
>>B.^2=                  % 矩阵元素乘方运算
      49     4
      1      0
>>A.^B=                  % 矩阵对应元素幂运算
      1      4
      1      1
>>C.'=                   % 矩阵转置
      1.0000+2.0000i,3.0000+1.0000i
      5.0000-2.0000i,1.0000+3.0000i
```

作为高性能的科学计算工具，MATLAB 对矩阵与向量提供强大的支持；但由于本书主要讲述使用 Simulink 需要具备的 MATLAB 的基础知识，因此对这部分内容仅作简单介绍，感兴趣的读者可以参考任何一本 MATLAB 参考书。

2.3　多项式表达与基本运算

Simulink 用于动态系统建模、仿真与分析时，将会大量使用多项式。许多系统的模型描述(如系统的传递函数)都需要使用多项式，并在多项式描述的基础上对系统进行仿真分析。本节将简单介绍 MATLAB 中的多项式表示及其基本运算。

1. 多项式的建立

在 MATLAB 中，n 阶多项式 $p(x)$ 由一个长度为 $n+1$ 的向量 $\boldsymbol{p}$ 所表示，向量 $\boldsymbol{p}$ 的元素为多项式的系数，且按照自变量 x 的降序排列。若 n 阶多项式为

$$p(x)=a_n x^n + a_{n-1}x^{n-1}+\cdots+a_2x^2+a_1x+a_0 \quad (其中\ a_n \neq 0)$$

则其在 MATLAB 中的表示方法为

$$\boldsymbol{p}=[a_n \quad a_{n-1} \quad \cdots \quad a_2 \quad a_1 \quad a_0]$$

注意，多项式中系数为 0 的项不能忽略，$\boldsymbol{p}$ 中相应元素应置为 0。如多项式 $p(x)=3x^3+2x+3$ 在 MATLAB 中应表示为

```
>>p=[3   0   2   3]
```

2. MATLAB 中多项式操作函数简介

(1) roots(p)：长度为 n 的向量，表示 n 阶多项式的根，即方程 p(x)=0 的根，可以为复数。

(2) conv(p，q)：表示多项式 p，q 的乘积，一般也指 p，q 的卷积。

(3) poly(A)：计算矩阵 A 的特征多项式向量。

(4) poly(p)：由长度为 n 的向量中的元素为根建立的多项式，结果是长度为 n+1 的向量。

(5) polyval(p，x)：若 x 为一数值，则计算多项式在 x 处的值；若 x 为向量，则计算多项式在 x 中每一元素处的值。

3. 举例分析

【例 2.4】 求多项式 $p(x)=3x^3+2x+3$ 的根。

解：在 MATLAB 的命令窗口中依次输入以下命令，即可求出其根。

```
>> p=[3    0    2    3];
>>rootp=roots(p)                              % rootp 为多项式的根
>>rootp =
       0.3911 + 1.0609i
       0.3911 - 1.0609i
      -0.7822
```

【例 2.5】 求多项式 $q(x)=2x+3$ 与 $p(x)=3x^3+2x+3$ 的乘积。

解：在 MATLAB 的命令窗口中依次输入以下命令，即可求出乘积。

```
>> p=[3    0    2    3];
>>q=[2    3];
>>prod=conv(q,p)                              % prod 为多项式 p，q 乘积多项式的系数
prod =
       6     9     4     12     9
```

即多项式 $q(x)$ 和 $p(x)$ 的乘积为多项式 $prod(x)=6x^4+9x^3+4x^2+12x+9$。

2.4 MATLAB 的基本绘图功能

MATLAB 作为高性能、交互式的科学计算工具，具有非常友好的图形界面，这使得 MATLAB 的应用非常广泛；同时 MATLAB 也提供了强大的绘图功能，这使得用户可以通过对 MATLAB 内置绘图函数的简单调用，便可迅速绘制出具有专业水平的图形。在利用 Simulink 进行动态系统仿真时，图形输出可以使设计者快速地对系统性能进行定性分析，故可大大缩短系统开发时间。

MATLAB 的图形系统是面向对象的。图形的要素，如坐标轴、标签、观察点等都是独立的图形对象。一般情况下，用户不需直接操作图形对象，只需调用绘图函数就可以得到理想的图形。通过本节的学习，用户能够快速掌握图形绘制技术。

1. 基本的二维图形绘制命令

(1) plot(x，y)：输出以向量 x 为横坐标，以向量 y 为纵坐标且按照 x，y 元素的顺序有序绘制的图形。x 与 y 必须具有相同长度。

(2) plot(y)：输出以向量 y 元素序号 m 为横坐标，以向量 y 对应元素 y_m 为纵坐标绘制的图形。

(3) plot(x1，y1，'str1'，x2，y2，'str2'，…)：用'str1'指定的方式，输出以 x1 为横坐标，y1 为纵坐标的图形。用'str2'指定的方式，输出以 x2 为横坐标，y2 为纵坐标的图形。若省略'str'，则 MATLAB 自动为每条曲线选择颜色与线型。'str'选项中的部分参数如表 2.1 所示。

表 2.1　plot 命令选项

颜色		线型			
'g'	绿色	'.'	粗点线	'--'	虚线
'y'	黄色	':'	点线	'-.'	点划线
'r'	红色	'*'	星线	'-'	实线
'b'	蓝色	'o'	圆圈	'+'	加号
'm'	品红色	'x'	叉	's'	方形
'y'	黄色	'd'	菱形	'p'	五角星
'k'	黑色	'^'	上三角	'h'	六角星

2. 简单的三维图形绘制命令

(1) plot3（x，y，z）：用向量 x、y 和 z 的相应点(x_i，y_i，z_i)进行有序绘制三维图形。向量 x，y，z 必须具有相同的长度。

(2) plot3(x1，y1，z1，'str1'，x2，y2，z2'str2'，…)：用'str1'指定的方式，对 x1，y1 和 z1 进行绘图；用'str2'指定的方式，对 x2，y2 和 z2 进行绘图；如果省略'str'，则 MATLAB 自动选择颜色与线型。

3. 图形绘制举例

【例 2.6】　用 MATLAB 绘制正弦函数在[0，2π]中的图形。

解： 在 MATLAB 命令行下输入

```
>> x=0:0.1:2*pi;
% pi 为 MATLAB 中默认的圆周率
>> y=sin(x);
>> plot(x, y, '*');
```

其中 x 为自变量，这里使用冒号表达式设定其取值步长为 0.1，取值范围为[0，2π]。用星号'*'输出图形，结果如图 2.2 所示。

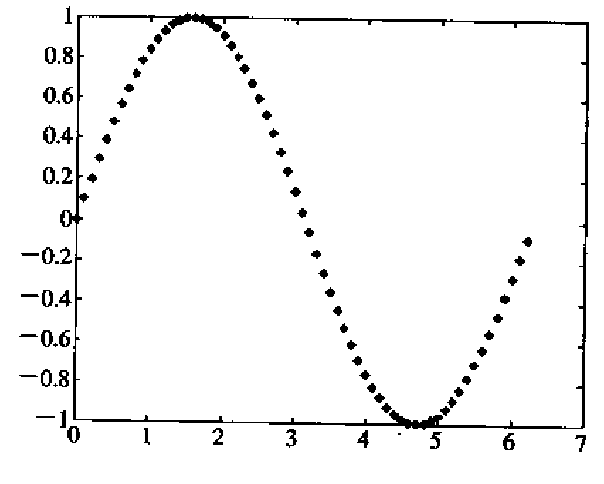


图 2.2　绘制正弦函数曲线

【例 2.7】　用 MATLAB 在同一图形窗口中绘制多项式 $q(x)=2x+3$ 与 $p(x)=3x^2+2x+3$ 的曲线，其中 $x\in[-2，5]$，要求分别用不同的线型与颜色表示。

解：在 MATLAB 命令行下输入

```
>> x=-2:0.1:5;
>> q=2*x+3;
>> p=3*x.^2+2*x+3;
>> plot(x,q,'r--',x,p,'b-')
```

结果如图 2.3 所示。直线（红色）表示多项式 $q(x)=2x+3$，曲线（蓝色）表示多项式 $p(x)=3x^2+2x+3$。

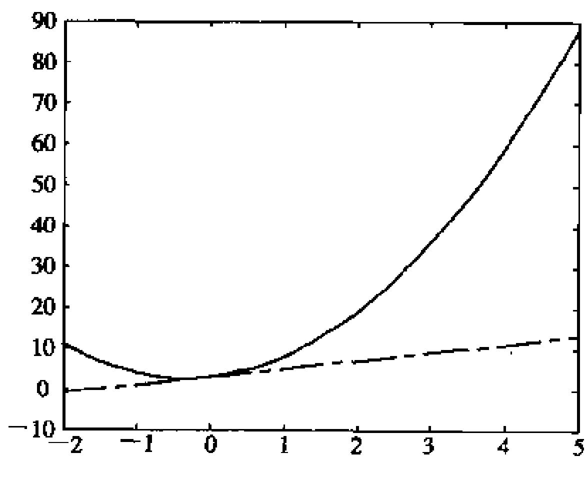


图 2.3 多项式绘制

4. 简单的图形控制命令

(1) clc：清除命令窗口。

(2) grid：自动在各个坐标轴上加上虚线型的网格。

(3) hold on：保持当前的图形，允许在当前图形状态下绘制其它图形，即在同一图形窗口中绘制多幅图形。

(4) hold off：释放当前图形窗口，绘制的下一幅图形将作为当前图形，即覆盖原来图形。这是 MATLAB 的缺省状态。

(5) hold：在 hold on 与 hold off 之间进行切换。

5. 简单的子图命令

(1) subplot(m,n,p)：将图形窗口分成 m 行 n 列的子窗口，序号为 p 的子窗口为当前窗口。子窗口的编号由上至下，由左至右。

(2) subplot：设置图形窗口为缺省模式，即 subplot(1，1，1)单窗口模式。

【例 2.8】 绘出在三维空间中的一个随机曲线。

解：在 MATLAB 命令行下输入

```
>> x=cumsum(rand(1,1000)-0.5);
>> y=cumsum(rand(1,1000)-0.4);
>> z=cumsum(rand(1,1000)-0.3);
>> plot3(x,y,z)
>> grid;
```

结果如图 2.4 所示。其中函数 cumsum(x)表示对向量 x 的各元素求累加和。Rand(m，n)

表示生成 m×n 的随机矩阵，且矩阵中所有元素服从[0,1]之间的均匀分布。grid 表示为各个坐标轴加上虚线型的网格。

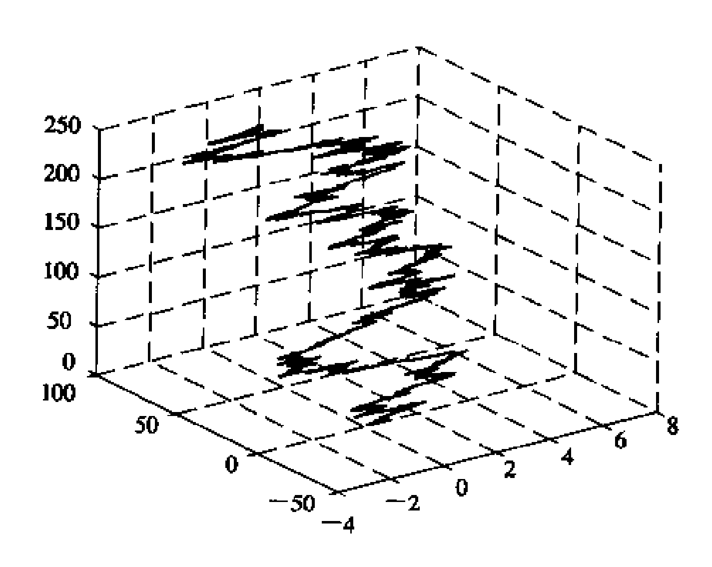


图 2.4　随机曲线绘制

【例 2.9】　在一个图形窗口的左侧子图中绘制函数 $y_1(x)=x^3-2x-3$，在右侧子图中绘制函数 $y_2(x)=x\sin(x)$，其中 $x\in[-3,3]$。

解：在 MATLAB 命令行下输入：

```
>> x=-3:0.1:3;
>> y1=x.^3-2*x-3;
>> y2=x.*sin(x);
>> subplot(1,2,1),plot(x,y1,'*'),grid
>> subplot(1,2,2),plot(x,y2,'-'),grid
```

结果如图 2.5 所示。

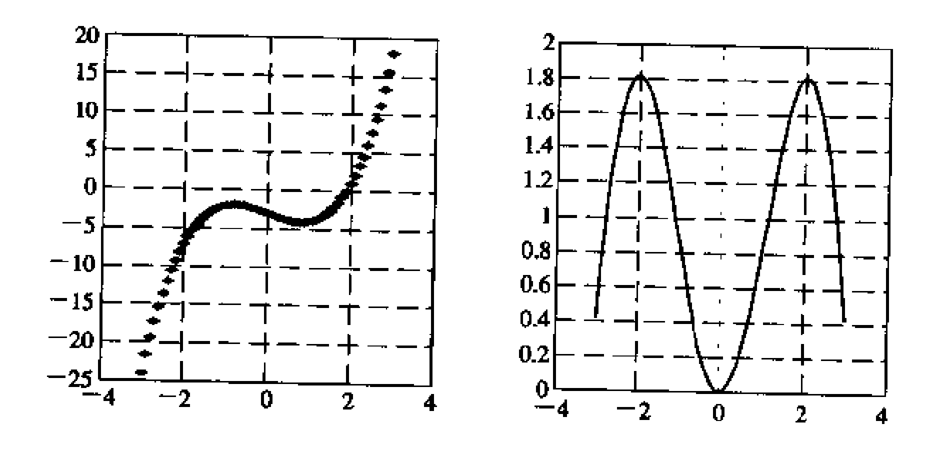


图 2.5　子图绘制

由此可见，MATLAB 的图形绘制功能非常强大，文中仅以几个简单的例子说明，读者可以进一步对生成的图形进行更低层操作(操作图形对象)，以便获得更好的效果，这里不再赘述。

2.5 M 文件与 MATLAB 函数

MATLAB 语言作为一种高级解释性语言，使用和调试起来都非常简单。利用 MATLAB 提供的丰富的数学函数，用户可以迅速展开自己的工作，编制验证自己的算法。本节首先介绍编写和调试 MATLAB 语言的工具：M File Editor。然后介绍 MATLAB 语言基本的语法结构和数据类型，如果读者已经有了 C、FORTRAN 或 PASCAL 等程序设计语言的基础，这部分的内容就显得非常简单。最后将介绍 MATLAB 语言的两种组织形式：MATLAB 脚本与 MATLAB 函数。

2.5.1 M 文件编辑器

“工欲善其事，必先利其器。”用户应首先熟悉一下最经常使用的 M 文件编辑器(M File Editor)。M 文件编辑器不仅仅是一个文字编辑器，它还具有一定的程序调试功能，虽然没有像 VC、BC 那样强大的调试能力，但对于调试一般不过于复杂的 MATLAB 程序已经足够了。在 MATLAB 命令行下输入

```
>>edit
```

则弹出如图 2.6 所示的 M 文件编辑器窗口。

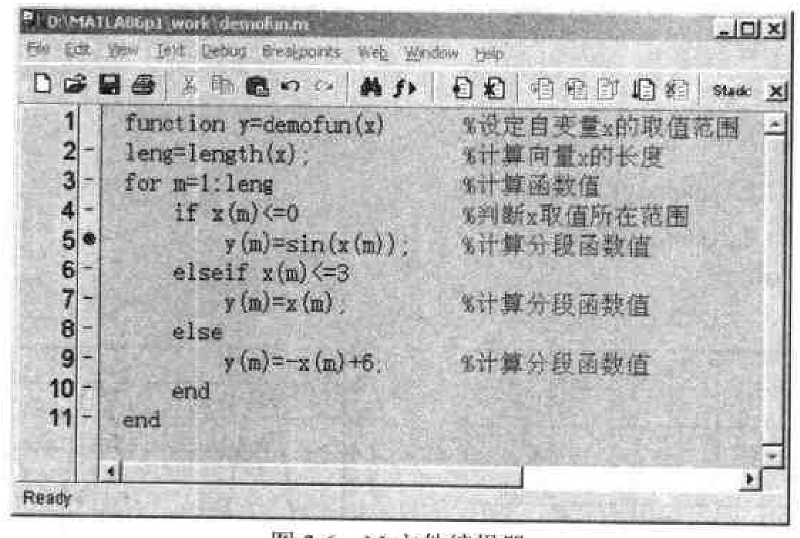


图 2.6 M 文件编辑器

1. 编辑功能

(1) 选择：与通常鼠标选择方法类似，但这样做其实并不方便。如果习惯了，使用 Shift+箭头键是一种更为方便的方法，熟练后根本就不需要再看键盘。选择括号中的表达式时有更方便的方法，将光标移入括号中，然后按快捷键 Ctrl+B 即可选中。M 文件编辑器中选中的文字用蓝底表示。

(2) 拷贝粘贴：没有比 Ctrl+C、Ctrl+V 键更方便的了，相信使用过 Windows 的人一定知道。

(3) 寻找替代：寻找字符串时用 Ctrl+F 键显然比用鼠标点击菜单方便。

(4) 查看函数：阅读大的程序常需要看看都有哪些函数并跳到感兴趣的函数位置，M 文

件编辑器没有为用户提供像VC或者BC那样全方位的程序浏览器，却提供了一个简单的函数查找快捷按钮，单击该按钮，会列出该M文件所有的函数。

(5) 注释：如果用户已经有了很长时间的编程经验而仍然使用Shift+5来输入%号，一定体会过其中的痛苦（忘了切换输入法状态时，就会变成中文字符集的百分号）。Ctrl+R和Ctrl+T把整行设置为注释或者取消注释，当需要注释一大段文字时，应该先选中这些文字。

(6) 缩进：良好的缩进格式为用户提供了清晰的程序结构。编程时应该使用不同的缩进量，以使程序显得错落有致。增加缩进量用Ctrl+]键，减少缩进量用Ctrl+[键。当一大段程序比较乱的时候，使用smart indent (聪明的缩进，快捷键Ctrl+I)也是一种很好的选择。

2. 调试功能

M程序调试器的热键设置和VC的设置有些类似，如果用户有其它语言的编程调试经验，则调试M程序显得相当简单。因为它没有指针的概念，这样就避免了一大类难以查找的错误。不过M程序可能会经常出现索引错误，如果设置了stop if error(Breakpoints菜单下)，则程序的执行会停在出错的位置，并在MATLAB命令行窗口显示出错信息。下面列出了一些常用的调试方法。

(1) 设置或清除断点：使用快捷键F12。

(2) 执行：使用快捷键F5。

(3) 单步执行：使用快捷键F10。

(4) step in：当遇见函数时，进入函数内部，使用快捷键F11。

(5) step out：执行流程跳出函数，使用快捷键Shift+F11。

(6) 执行到光标所在位置：非常遗憾这项功能没有快捷键，只能使用菜单来完成这样的功能。不过用户可以先用F12设置断点，再用F5执行到断点位置的方法实现上述功能。

(7) 观察变量或表达式的值：将鼠标放在要观察的变量上停留片刻，就会显示出变量的值，当矩阵太大时，只显示矩阵的维数。用户在MATLAB调试命令行下（K>>）输入变量或表达式后，系统就会列出它们的值。

(8) 退出调试模式：没有设置快捷键，使用菜单或者快捷按钮来完成。

2.5.2 MATLAB语言的语法

1. 注释

MATLAB中用百分号%表示其后为程序注释(实际上在前面已经碰到了这样的注释)。编写M程序和编写其它程序一样应该养成良好的程序注释习惯。除了程序间的注释，编写M文件时还应该在文件头说明该程序的功能和使用方法，使用Help命令看到的帮助信息正是这些在文件头的注释。

2. 赋值语句

在MATLAB中，赋值语句的基本语法结构为

variablename=value;

3. 逻辑表达式

在MATLAB中，逻辑表达式的基本语法结构为

logicalvalue＝variable1 关系运算符 varialble2;

logicalvalue＝logical expression 1 逻辑运算符 logical expression 2;

其中关系运算符有= =(等于)、～=(不等于)、>(大于)、<(小于)、>=(不小于)、<=(不大于)等。逻辑运算符有&(逻辑与)、|(逻辑或)和~(逻辑非)等。

4. 条件控制语句

MATLAB 中由 if 语句进行判断，其基本语法结构为

```
if  逻辑表达式
    语句集合
end
```

在 if 与逻辑表达式之间必须有一个空格；当逻辑表达式值为真时，执行语句集合中的语句；这里语句集合可以是 MATLAB 中的单独命令，也可以是由逗号、分号隔开的语句集合或 return 语句。

对于简单的语句也可以写成下面的形式：

```
if 逻辑表达式，语句集合，end
```

此外，if 语句还可以与 elseif、else 组合成更为复杂的控制语句，其语法格式如下：

```
if  逻辑表达式
    语句集合 1
else
    语句集合 2
end
```

说明：当逻辑表达式值为真时，执行语句集合 1；否则执行语句集合 2。

```
if  逻辑表达式 1
    语句集合 1
elseif 逻辑表达式 2
    语句集合 2
end
```

说明：当逻辑表达式 1 为真时，执行语句集合 1；若逻辑表达式 1 为假且逻辑表达式 2 为真，则执行语句集合 2。

另外，if 语句之间可以进行多级嵌套，以完成更为复杂的条件控制。其语法格式如下：

```
if  逻辑表达式 1
        ...
        if 逻辑表达式 2
            语句集合 1
        else
            语句集合 2
        end
        ...
else
        ...
        if 逻辑表达式 3
           语句集合 3
```

```
        elseif 逻辑表达式 4
            语句集合 4
        else
            语句集合 5
        end
        ...
end
```

另外一种条件语句是 switch-case 语句，其语法格式如下：

```
switch 逻辑表达式（标量或字符串）
case value1
        语句集合 1
case value2
        语句集合 2
...
otherwise
        语句集合
end
```

说明：计算逻辑表达式之值，若其与 value1 匹配，则执行语句集合 1；若与 value2 匹配，则执行语句集合 2；若没有匹配的，则执行 otherwise 中的语句集合。

5. 循环语句

MATLAB 中实现循环的语句有两种：for 语句与 while 语句，以实现某些语句的循环执行。for 语句语法格式如下：

```
for 变量=表达式
      语句集合
end
```

说明：这里的变量是循环变量。在表达式中应给出循环变量的初始值、步长和终值，如 i=1:2:100。这个步长可为负数或单位值。如果步长为单位值 1，则可以略去不写。

for 语句同样可以嵌套使用，其语法如下：

```
for 变量 1=表达式 1
...
    for 变量 2=表达式 2
        语句集合
    end
...
end
```

另一种实现循环的 while 语句的语法格式如下：

```
while 逻辑表达式
    语句集合
end
```

说明：当逻辑表达式值为真时，程序一直循环执行语句集合，直到逻辑表达式为假时终止循环。while 语句也可以进行类似的嵌套使用，这里不再赘述。

2.5.3 MATLAB 脚本文件与 M 函数

MATLAB 中有两种 M 文件：一种称为脚本文件（类似于批处理语句），另一种是 M 函数（类似于函数的概念）。

1. 脚本文件

脚本文件是由一系列 MATLAB 的命令、内置函数以及 M 文件等构成的文件，它可以由一般的编辑器进行编制，其结果保存在相应的 M 文件中。M 脚本文件的实质为命令的集合，在 MATLAB 中执行 M 脚本文件时，MATLAB 从文件中读取命令执行，完成用户的工作。

一般习惯于使用 MATLAB 的编辑器编制 M 文件。打开 MATLAB 编辑器，新建 M 脚本文件，保存时系统会自动将文件保存成*.m 文件。然后便可以在 MATLAB 命令窗口或其它 M 文件中使用。其特点是按照脚本中语句的顺序执行，生成的变量放在当前的工作区之中（如果从命令行运行，则放在基本工作区）。

【例 2.10】 编写一个 M 文件绘制函数 $y(x)=\begin{cases}\sin x, & x\leqslant 0\\ x, & 0<x\leqslant 3\\ -x+6, & x>3\end{cases}$ 在区间$[-6,6]$中的图形。

解： 在 MATLAB 命令行下输入 edit 命令以打开 M 文件编辑器，输入以下程序：

```
x=-6:0.1:6;                      % 设定自变量 x 的取值范围
leng=length(x);                  % 计算向量 x 的长度
for m=1:leng                     % 计算函数值
    if x(m)<=0                   % 判断 x 取值所在范围
        y(m)=sin(x(m));          % 计算分段函数值
    elseif x(m)<=3
        y(m)=x(m);               % 计算分段函数值
    else
        y(m)=-x(m)+6;            % 计算分段函数值
    end
end
plot(x,y,'*'), grid;             %绘制函数曲线
```

将其存盘为 demomfile1.m（该文件就是一个 MATLAB 脚本文件），然后在 MATLAB 命令行下输入：

```
>>demomfile1
```

则生成如图 2.7 所示的函数曲线。

本例目的在于说明 M 脚本文件的编写技术，以及如何使用前面所讲述的 MATLAB 语言的条件判断与循环语句。由此可见使用 MATLAB 语言进行程序设计简单而又快速。

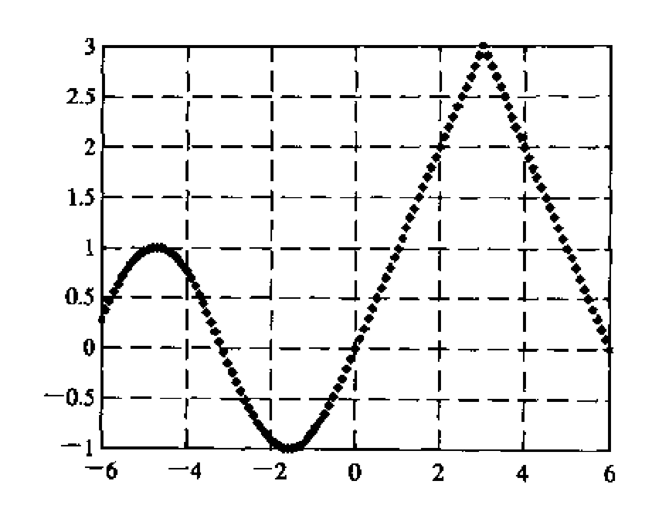


图 2.7 使用 M 文件绘制函数图形

2. M 函数

MATLAB 的函数与脚本不同，M 函数的第一行为关键字 function，函数第一次执行时将生成内存代码，生成的变量放在函数的工作区。在 MATLAB 中有大量的内置函数及大量的工具箱函数，使用它们可以完成大部分的工作；但由于不同的用户有不同的需要，MATLAB 允许用户开发自己的专用或通用函数，以扩展 MATLAB 的函数应用。这里仅简单介绍一下 M 函数的编制与使用方法。这对理解后面的 S-函数有很重要的作用。

(1) M 函数的第一行必须包含 function，普通的 M 文件没有这种要求。

(2) 在 function 后面必须声明函数名、输入变量（输入参数）与输出变量（输出参数），如 function outvar=function_name(inputvar)。

(3) M 函数可以有零个、一个或多个输入或输出。

(4) M 函数的调用方式为：outvar=function_name(inputvar)。

(5) M 函数文件名须和函数名 function_name 相同，调用时函数的输入与输出变量名称不需要和函数定义中的变量相同。

(6) M 函数的注释用%开始的行表示，help function_name 显示的是紧接第一行之后的注释。

MATLAB 允许将多个函数写在同一个 M 文件中，其中第一个函数是 M 文件的主函数，M 文件名必须为主函数的名字。其余的函数均为子函数，并受到其它函数的调用。因此，用户可以书写具有模块化特色的 MATLAB 函数，但是要注意以下几点：

(1) 所有的子函数只能在同一 M 文件下调用。

(2) 每个子函数都有自己单独的工作区，必须由调用函数传递合适的参数。

(3) 当子函数调用结束后，子函数的工作区将被清空。

函数/子函数的概念将在 S-函数中得到应用。在 S-函数编写中将大量使用子函数技术。这里不再作过多介绍。

【例 2.11】 编写一个通用的 M 函数求取【例 2.10】中函数在任意点的值，并绘制函数在区间[−6,6]中的图形。

解：(1) 编写函数 demofun 并将其存储在同名 M 文件 demofun.m 中。

```
function y=demofun(x)              % M 函数定义
leng=length(x);                    % 计算向量 x 的长度
for m=1:leng                       % 计算函数值
    if x(m)<=0                     % 判断 x 取值所在范围
        y(m)=sin(x(m));            % 计算分段函数值
    elseif x(m)<=3
        y(m)=x(m);                 % 计算分段函数值
    else
        y(m)=-x(m)+6;              % 计算分段函数值
    end
end
```

(2) 编写 M 脚本文件 demofile2.m，绘制函数曲线或在命令行下输入下列命令：

```
x=-6:0.1:6;              % 设定 x 的取值范围
y=demofun(x);            % 调用函数 demofun.m 求值
plot(x,y), grid;         % 输出图形
```

结果如图 2.8 所示，与前面输出图形一致。

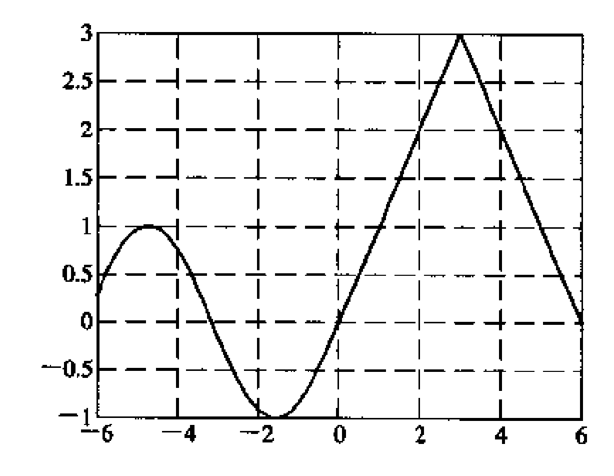


图 2.8 使用子函数绘制图形

M 函数可以较好地将具有一定功能的脚本文件进行封装，这样有利于程序的阅读与修改。这时可以在 MATLAB 命令窗口中显示工作区中的变量，输入以下命令：

```
>>whos
  Name      Size      Bytes     Class
  x         1×121     968       double array
  y         1×121     968       double array
Grand total is 242 elements using 1936 bytes
```

可见，在 MATLAB 窗口中只有脚本文件 demofile2.m 的工作区变量，而函数 demofun.m 的工作区在函数调用完毕后自动被清除。这便是二者区别之一。

2.6 MATLAB 的单元与结构体

1. 字符串数据

MATLAB 作为高性能的科学计算平台，不仅提供高精度的数值计算功能，而且还提供对多种数据类型的支持。如 double 类型表示双精度浮点数，char 表示字符，unit8 表示无符号 8 位整型数等等。除此之外，MATLAB 还提供对字符串的支持，在 MATLAB 中字符串由单引号来定义，如

```
Strname='Simulation'    % 表示 Strname 为一字符串，其值为 Simulation
```

进而可以定义字符（串）矩阵，它与定义普通矩阵类似。

2. 单元矩阵

在前面所提到的矩阵与向量中，矩阵之中所有元素的数据类型均为单一的类型。MATLAB 支持复合数据类型的矩阵与向量，这是由一个特殊的矩阵实现的，它就是单元矩阵(Cell 类型的矩阵)。在有些书中，单元矩阵也称为细胞矩阵或细胞数组。单元矩阵不同位置可以有不同的数据类型。这使其与仅由数字构成的矩阵和仅由字符构成的矩阵完全不同。单元矩阵可以是一维的（即单元向量），也可以是多维的，其引用方式等与普通的矩阵类似。但其生成方式和一些操作与普通矩阵不尽相同。单元矩阵的生成方式有如下三种：

(1) 使用花括号{ }直接生成，这与普通矩阵使用中括号[ ]生成方法一致。例如

```
>>cellmatrix={'xidian', 'press', 20; 'xian', 15.21, 'university'};
```

(2) 直接对单元矩阵中的每一单元分别进行赋值，如

```
>>cellname{1}='MATLAB';
>>cellname{2}=20.23;
```

(3) 通过 MATLAB 中单元矩阵的创建命令 cell 创建合适的矩阵。cell 的使用方法如下：

```
>>cellname=cell (m, n)    % 表示创建一个 m×n 的单元矩阵
```

MATLAB 中提供了其它一些对单元矩阵的操作命令，这里不对其进行更多介绍。

3. 结构体

如今的程序设计语言中，大都提供了对结构体变量的支持；MATLAB 同样支持结构体变量，而且其生成与使用都非常容易、直观。结构体是一个很有用的某些具有某种相关性记录的集合体，它使一系列相关记录集合到一个统一的结构之中，从而使这些记录能够被有效地管理、组织与引用。

在 MATLAB 中，结构体是按照域的方式生成与存储结构体中的每个记录；一个域中可以包括任何 MATLAB 支持的数据类型，如双精度数值、字符、单元矩阵及结构等类型。下面简单介绍结构体的生成与引用。

1) 结构体生成

结构体生成方式：struct_name(record_number).field_name=data;

如某个班级学生花名册的建立：

```
>>student(1).name='Li Yang';
>>student(1).number='0134';
```

```
>>student(2).name='Ma Lei';
>>student(2).number='0135';
...
>>student(33).name='Yao Hui';
>>student(33).number='0166'
```

student 是具有 33 个结构变量的向量，表示某个班级所有 33 个同学的姓名与学号。每一个记录对应一个学生的姓名与学号。由此可见，在 MATLAB 中建立结构体是不费吹灰之力的。

2) 结构体引用

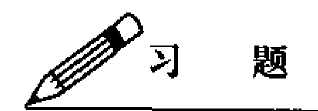
在 MATLAB 中对结构体变量的引用也很简单，如对上述学生花名册中的第二个学生记录的引用如下：

```
>>Name=student(2).name;
>>Number=student(2).number;
```

其结果为

```
Name=
    Ma Lei
Number=
    0134
```

在 Simulink 中可以使用结构体数据，以包括信号更多的信息（如信号的名称、信号相应的时刻等）。



习　题

1. 求多项式 $p(x)=2x^3+x+3$ 的根。

提示：使用 roots 命令。

2. 绘制函数 $f(x)=\begin{cases}\sin x, & x>5\\ 3+x, & 0<x\leqslant 5\\ x^2, & x\leqslant 0\end{cases}$ 在 $[-10, 10]$ 之间的图形。

提示：使用 plot 命令。

3. 编写 M 文件绘制题 2 中函数在不同区间的图形，其区间分别为 $[-1, 1]$、$[3, 5]$。

提示：参考【例 2.10】。

第3章 动态系统模型及其 Simulink 表示

内容概要

- 简单系统模型及表示
- 离散系统模型及表示
- 连续系统模型及表示
- 混杂系统模型及表示

任何一个系统，无论其结构如何复杂，系统的输入与输出都具有某种关系。根据不同的输入输出关系，可以将系统分成不同的类型。在使用 Simulink 进行动态系统仿真之前，读者最好能够了解有关动态系统的一些基本知识。这对使用 Simulink 进行系统分析与仿真有着很大的帮助。本章将简单介绍各种类型动态系统的基本知识，以作为使用 Simulink 进行动态系统仿真分析的基础。

3.1　简单系统模型及表示

3.1.1　简单系统的基本概念

不同系统具有不同数量的输入与输出；一般来说，输入输出数目越多，系统越复杂。最简单的系统一般只有一个输入与一个输出，而且任意时刻的输出只与当前时刻的输入有关。本节首先介绍简单系统的基本概念以及简单系统的 Simulink 表示。

【定义 3.1】　**简单系统**。对于满足下列条件的系统，我们称之为简单系统：

(1) 系统某一时刻的输出直接且唯一依赖于该时刻的输入量。

(2) 系统对同样的输入，其输出响应不随时间的变化而变化。

(3) 系统中不存在输入的状态量，所谓的状态量是指系统输入的微分项（即输入的导数项）。

设简单系统的输入为 x，系统输出为 y，x 可以具有不同的物理含义。对于任何系统，都可以将它视为对输入变量 x 的某种变换，因此可以用 $T[\,]$ 表示任意一个系统，即

$$y=T[x]$$

对于简单系统，x 一般为时间变量或其它的物理变量，并具有一定的输入范围。系统输出变量 y 仅与 x 的当前值相关，从数学的角度来看，y 是 x 的一个函数，给出一个 x 值，便有一个 y 值与之对应。

【例 3.1】　对于如下的一个系统：

$$y=\begin{cases}u^2, & t\in[0,1]\\ u^{\frac{1}{2}}, & t>1\end{cases}$$

其中u为系统的输入变量，t为时间变量，y为系统的输出变量。输入变量$u=t$。很显然，此系统服从简单系统的条件，为一简单系统。系统输出仅由系统当前时刻的输入决定。

3.1.2 简单系统的描述方式

一般来讲，简单系统都可以采用代数方程与逻辑结构相结合的方式进行描述。

1. 代数方程

采用数学方程对简单系统进行描述，可以很容易由系统输入求出系统输出，并且由此可方便地对系统进行定量分析。

2. 逻辑结构

一般来说，系统输入都有一定的范围。对于不同范围的输入，系统输出与输入之间遵从不同的关系。由系统的逻辑结构可以很容易了解系统的基本概况。

3.1.3 简单系统的 Simulink 描述

本章主要介绍动态系统的基本知识，为使用 Simulink 进行系统仿真打下基础。因此这里并不准备建立系统的 Simulink 模型，而是采用编写 M 脚本文件的方式对系统进行描述并进行简单的仿真。下面以【例 3.1】中的简单系统为例，说明在 Simulink 中如何对简单系统进行描述。

对于【例 3.1】中的简单系统，编写如下的 systemdemo1.m 脚本文件进行描述与分析。

```
% systemdemo1.m 文件
u=0:0.1:10;                    % 设定系统输入范围与仿真步长
leng=length(u);                % 计算系统输入序列长度
for i=1:leng                   % 计算系统输出序列
    if u(i)<=1                 % 逻辑判断
        y(i)=u(i).^2;
    else
        y(i)=sqrt(u(i));
    end
end
plot(u,y);grid;   % 绘制系统仿真结果
```

图 3.1 所示为此简单系统在[0，10]时间之内的输入输出关系图，其实就是系统在此范围内的仿真结果。修改此 M 脚本文件，不断改变系统输入的时间范围与系统输入的仿真步长，便可以完成系统的仿真分析，并绘制不同的仿真结果。当然，采用 Simulink 模型可以进行交互式的仿真。在第 5 章中将使用 Simulink 对系统进行交

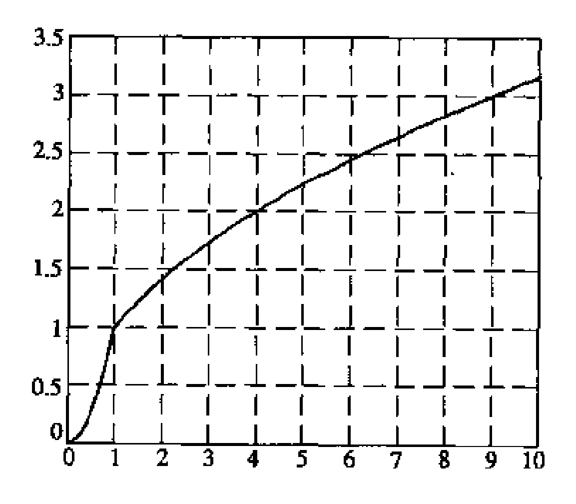


图 3.1 简单系统的输入输出关系图

方式的仿真分析。这里仅从概念上讲述系统的基本知识。

3.2　离散系统模型及表示

3.2.1　离散系统的基本概念

前面所涉及到的系统中，无论是系统的输入还是系统的输出均是连续的变量，在这里连续指的是系统的输入与输出均在时间变量上连续取值（与数学上函数连续概念并不相同）。本节将简单介绍离散系统的基本概念，系统的描述与简单仿真。

所谓离散系统，是指系统的输入与输出仅在离散的时间上取值，而且离散的时间具有相同的时间间隔。下面给出离散系统更全面的定义。

【定义 3.2】**离散系统**。凡是满足如下条件的系统均为离散系统：

(1) 系统每隔固定的时间间隔才“更新”一次，即系统的输入与输出每隔固定的时间间隔便改变一次。固定的时间间隔称为系统的“采样”时间。

(2) 系统的输出依赖于系统当前的输入、以往的输入与输出，即系统的输出是它们的某种函数。

(3) 离散系统具有离散的状态。其中状态指的是系统前一时刻的输出量。

3.2.2　离散系统的数学描述

前面给出了离散系统的定义，这里给出离散系统的数学描述。设系统输入变量为 $u(nT_s), n=0,\ 1,\ 2\cdots\cdots$，其中 T_s 为系统的采样时间，n 为采样时刻。显然，系统的输入变量每隔固定的时间间隔改变一次。由于 T_s 为一固定的值，因而系统输入 $u(nT_s)$ 常被简记为 $u(n)$。设系统输出为 $y(nT_s), n=0,\ 1,\ 2\cdots\cdots$，同样也可简记为 $y(n)$。由离散系统的定义可知，其数学描述应为

$$y(n)=f\big(u(n)\quad u(n-1)\cdots\cdots;\quad y(n-1)\quad y(n-2)\cdots\cdots;\big)$$

注意：对于离散系统，由于系统的输出不仅和系统当前的输入有关，而且还和系统以往的输入与输出相关；而对于一般的系统而言，系统均是从某一时刻开始运行的。因此，在离散系统中，系统初始状态的确定是非常关键的。对于同样的一个系统，如果系统的初始状态不相同，那么系统的运行情况有可能会完全不同，甚至相反。本书并不准备详细说明这一点，仅仅提醒读者注意：要给离散系统设定合适的初值。

【例 3.2】　对于如下的离散系统模型：

$$y(n)=u(n)^2+2u(n-1)+3y(n-1)$$

其中系统的初始状态为 $y(0)=3$，系统输入为 $u(n)=2n, n=0,\ 1,\ 2\cdots\cdots$，则系统在时刻 0，1，2……的输出分别为

$$\begin{aligned}&y(0)=3\\&y(1)=u^2(1)+2u(0)+3y(0)=4+0+9=13\\&y(2)=u^2(2)+2u(1)+3y(1)=16+4+39=59\\&\cdots\cdots\end{aligned}$$

离散系统除了采用一般的数学描述方式之外，还可以采用差分方程进行描述。使用差分方程描述方程形式如下：

设系统的状态变量为 $x(n)$，离散系统差分方程由以下两个方程构成：

$$\text{状态更新方程：}\ x(n+1)=f_d(x(n),u(n),n)$$

$$\text{系统输出方程：}\ y(n)=g(x(n),u(n),n)$$

3.2.3 离散系统的 Simulink 描述

这里以【例 3.2】中的离散系统为例，说明如何利用 Simulink 对离散系统进行描述，并在此基础上对系统进行简单的分析。与前面相类似，此处并不建立系统的 Simulink 模型进行仿真，而是编写 M 脚本文件从原理上对离散系统进行说明，并说明离散系统与连续系统的区别之处。

编写脚本文件 systemdemo2.m 对【例 3.2】中的离散系统进行描述分析。

```
%systemdemo2.m 文件
y(1)=3;
                % 表示离散系统初始状态为 3
                %由于 MATLAB 中数组下标从 1 开始，这里 y(1)相当于上文中的 y(0)=3，下同
u(1)=0;                              % 表示离散系统初始输入为 0
for i=2:11                           % 设定离散系统输入范围为时刻 0 到时刻 10
    u(i)=2*i;                        % 离散系统输入向量
    y(i)=u(i).^2+2*u(i-1)+3*y(i-1);  % 离散系统输出向量
end
plot(u,y);grid;                      % 绘制系统仿真结果
```

系统从时刻 0 到时刻 10 的输入与输出的关系如图 3.2 所示。其中横坐标表示离散系统的输入向量，而纵坐标表示离散系统的输出向量。说明：这里并没有指定离散系统的采样时间，而仅仅举例说明离散系统的求解分析。在实际的系统中，必须指定系统的采样时间，只有这样才能获得离散系统真正的动态性能。

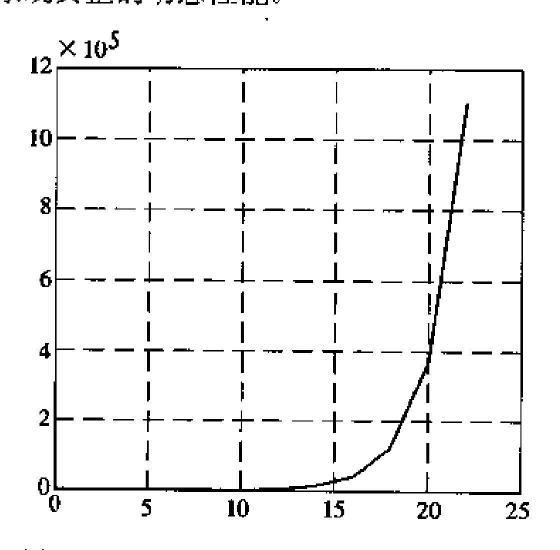


图 3.2 【例 3.2】中离散系统的输入输出关系

由此可见，此离散系统为非线性离散系统。

下面着重介绍线性离散系统的概念及其变换域描述。

3.2.4 线性离散系统

对于任何系统而言，系统的描述都可以采用抽象的数学形式来进行描述。这是因为任何系统都可以被看作是输入到输出的某种变换。例如，离散系统可以由下述的变换进行描述：

$$y(n)=T\{u(n)\},\ n=0,1,2,3\cdots\cdots$$

在离散系统之中，线性离散系统具有重要的地位。下面对线性离散系统进行简单的介绍。在此之前，读者需要理解如下的两个概念：

(1) 齐次性：若对于离散系统 $y(n)=T\{u(n)\}, n=0,1,2,3\cdots\cdots$，如果对任意的输入 $u(n)$ 与给定的任意常数 α，恒有

$$T\{\alpha u(n)\}=\alpha T\{u(n)\}$$

则称系统满足齐次性。

(2) 叠加性：如果系统对于输入 $u_1(n)$ 和 $u_2(n)$，输出分别为 $y_1(n)$ 和 $y_2(n)$，恒有

$$T\{u_1(n)+u_2(n)\}=T\{u_1(n)\}+T\{u_2(n)\}$$

则称系统满足叠加性。

【定义 3.3】**线性离散系统**。当离散系统同时满足齐次性与叠加性时，即

$$T\{\alpha u_1(n)+\beta u_2(n)\}=\alpha T\{u_1(n)\}+\beta T\{u_2(n)\}$$

则称此离散系统为线性离散系统。

例如，对于如下的离散系统：

$$y(n)=u(n)^2+2u(n-1)$$

由于 $T\{\alpha u(n)\}=\alpha^2u^2(n)+2\alpha u(n-1)$，而 $\alpha T\{u(n)\}=\alpha u^2(n)+2\alpha u(n-1)$，显然 $T\{\alpha u(n)\}\neq\alpha T\{u(n)\}$，系统不满足齐次性。故此系统不是线性离散系统。而对于下面的离散系统：

$$y(n)=u(n)+u(n-1)$$

由于 $T\{\alpha u(n)\}=\alpha u(n)+\alpha u(n-1)=\alpha T\{u(n)\}=\alpha u(n)+\alpha u(n-1)$，故系统满足齐次性，且有

$$\begin{aligned}T\{u_1(n)+u_2(n)\}&=u_1(n)+u_2(n)+u_1(n-1)+u_2(n-1)\\&=\{u_1(n)+u_1(n-1)\}+\{u_2(n)+u_2(n-1)\}\\&=T\{u_1(n)\}+T\{u_2(n)\}\end{aligned}$$

故系统满足叠加性，所以此系统为线性离散系统。

3.2.5 线性离散系统的数学描述

对于线性离散系统来说，可以使用最一般的方式对其进行描述，如采用如下的数学方程进行描述：

$$y(n)=f(u(n)\quad u(n-1)\cdots\cdots;\quad y(n-1)\quad y(n-2)\cdots\cdots;)$$

或采用差分方程进行描述：

状态更新方程：$x(n+1)=f_d(x(n),u(n),n)$

系统输出方程：$y(n)=g(x(n),u(n),n)$

除了使用一般的方式描述线性离散系统之外，针对线性离散系统本身的特点，经常使用 Z 变换来描述线性离散系统。Z 变换是对离散信号进行分析的一个强有力的工具，尤其是对线性离散系统。Z 变换有丰富的内容，但由于本书的目的主要是对各种实际的系统进行 Simulink 仿真，故在此仅简单介绍线性离散系统的 Z 变换域描述以及 MATLAB 中一些比较常用的对线性离散系统进行分析的函数。

Z 变换的含义：对于一个离散信号 $u(n)$，其 Z 变换为 $U(z)=\sum_{k=-\infty}^{\infty}u(k)z^{-k}$。一般来说，离散信号 $u(n)$ 的起始时间往往大于 0，这时它的 Z 变换为 $U(z)=\sum_{k=0}^{\infty}u(k)z^{-k}$。一般可以将其简记为 $U(z)=Z\{u(n)\}$。由离散信号的 Z 变换确定离散信号的过程为 Z 变换的逆变换，一般简记为 $u(n)=Z^{-1}\{U(z)\}$。

Z 变换具有多种不同的性质，这里仅介绍 Z 变换的如下两个性质：

(1) 线性性。即对于离散信号 $u_1(n)$ 和 $u_2(n)$，设它们的 Z 变换分别为 $U_1(z)$ 与 $U_2(z)$，所谓 Z 变换的线性性指的是 Z 变换满足下面的关系：

$$Z\{\alpha u_1(n)+\beta u_2(n)\}=\alpha Z\{u_1(n)\}+\beta Z\{u_2(n)\}$$

(2) 设离散信号 $u(n)$ 的 Z 变换为 $U(z)$，则 $u(n-1)$ 的 Z 变换为 $z^{-1}U(z)$。

下面举例说明如何使用 Z 变换域描述方式对线性离散系统进行描述。

【例 3.3】 对于如下的线性离散系统：

$$y(n)=u(n)+2u(n-1)+3y(n-1)$$

同时对等式两边进行 Z 变换，则有 $Y(z)=U(z)+2z^{-1}U(z)+3z^{-1}Y(z)$。一般在系统分析中，往往对系统输出与系统输入的比值比较关心，将此式化成分式的形式，有

$$\frac{Y(z)}{U(z)}=\frac{1+2z^{-1}}{1-3z^{-1}}=\frac{z+2}{z-3}$$

对于一般的线性离散系统，其 Z 变换域描述方式与此例相似：不妨设线性离散系统的 Z

变换描述形式为

$$\frac{Y(z)}{U(z)}=\frac{n_0+n_1z^{-1}+n_2z^{-2}}{d_0+d_1z^{-1}}$$

在对系统进行描述分析时，此种形式的描述称之为滤波器描述。对上式进行等价变换，可以得到系统的传递函数描述线性系统最常用的一种描述方式：

$$\frac{Y(z)}{U(z)}=\frac{n_0z^2+n_1z+n_2}{d_0z^2+d_1z}$$

还可以得到系统的零极点描述：

$$\frac{Y(z)}{U(z)}=k\frac{(z-z_1)(z-z_2)}{z(z-p_1)}$$

其中，z_1、z_2 为系统的零点，p_1 为系统极点，k 为系统增益。

3.2.6 线性离散系统的 Simulink 描述

在线性离散系统仿真分析中，很少直接使用数学方程进行系统描述，一般都采用线性离散系统的滤波器模型、传递函数模型，以及状态空间模型进行系统描述。本部分介绍线性离散系统的各种描述模型的 Simulink 实现。线性离散系统的描述方式有如下四种形式：

(1) 线性离散系统的滤波器模型：在 Simulink 中，滤波器表示为 num=[n0 n1 n2]; den=[d0 d1]；其中 num 表示 Z 变换域分式的分子系数向量，den 为分母系数向量。

(2) 线性离散系统的传递函数模型：在 Simulink 中，系统的传递函数表示为 num=[n0 n1 n2]; den=[d0 d1];

(3) 线性离散系统的零极点模型：在 Simulink 中，系统零极点表示为 gain=K; zeros=[z1, z2]; poles=[0, p1];

(4) 线性离散系统的状态空间模型：在 Simulink 中，设系统差分方程为如下形式：$x(n+1)=\boldsymbol{F}x(n)+\boldsymbol{G}u(n)$；$y(n)=\boldsymbol{C}x(n)+\boldsymbol{D}u(n)$。其中 $x(n)$, $u(n)$, $y(n)$ 分别为线性离散系统的状态变量、输入向量、输出向量。$\boldsymbol{F}$，$\boldsymbol{G}$，$\boldsymbol{C}$，$\boldsymbol{D}$ 分别为变换矩阵。在 Simulink 中，其表示很简单，只需要输入相应的变换矩阵 $\boldsymbol{F}$，$\boldsymbol{G}$，$\boldsymbol{C}$，$\boldsymbol{D}$ 即可。

一般来说，在不同的条件下使用不同的系统模型。在 MATLAB 中可以对系统的不同描述进行相互转化，并使用不同的命令对系统进行简单的频率响应分析。下面举例说明线性离散系统的简单频率分析。

【例 3.4】 对于如下的线性离散系统：

$$\frac{Y(z)}{U(z)}=\frac{2z^2-z-5}{z^3+3z^2+6z+2}$$

在 MATLAB 中输入下面的语句，可以绘制出此系统的 Bode 图：

```
>>num=[2  -1  -5];
>>den=[1  3  6  2];
>>dbode(num,den,1)
```

```
>>grid;
```

此离散系统的 Bode 图如图 3.3 所示。

当然也可以用下面的语句求出系统的幅值与相位而不绘制图形：

```
>>[mag,phase]=dbode(num,den,1);
```

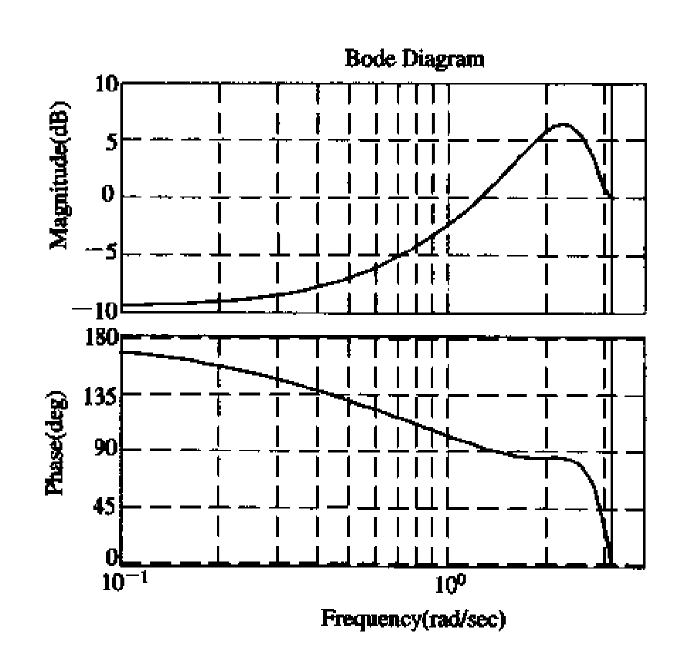


图 3.3 线性离散系统的 Bode 图

此外，在 MATLAB 中，离散系统的不同描述模型之间可以进行相互转化。这里给出几个比较常用的函数：

```
[zeros,poles,k]=tf2zp(num,den)        % 将系统传递函数模型转化为零极点模型
[num,den]=zp2tf(zeros,poles,k)
    % 将系统零极点模型转化为传递函数模型。其中 num，den 分别为系统的传递函数表
    % 示；zeros，poles，k 为系统的零极点模型
```

至于线性离散系统的状态空间模型描述，这里不再介绍，感兴趣的读者可以参考其它有关的书籍。这里给出它与传递函数模型、零极点模型相互转化的函数命令：

```
[zeros,poles,k]=ss2zp(F,G,C,D)     % 将系统状态空间模型转化为零极点模型
[F,G,C,D]=zp2ss(zeros,poles,k)     % 将系统零极点模型转化为状态空间模型
[num,den]=ss2tf(F,G,C,D)           % 将系统状态空间模型转化为传递函数模型
[F,G,C,D]=tf2ss(num,den)           % 将系统传递函数模型转化为状态空间模型
```

【例 3.5】 以线性离散系统 $\dfrac{Y(z)}{U(z)}=\dfrac{2z^2-z-5}{z^3+3z^2+6z+2}$ 为例说明系统模型的转化。

解：将传递函数模型转化为零极点模型：

```
>> num=[2  -1  -5];
>>den=[1  3  6  2];
>> [zeros,poles,k]=tf2zp(num,den)
```

结果为

```
>> zeros =
     1.8508
    -1.3508
>>poles =
   -1.2980 + 1.8073i
   -1.2980 - 1.8073i
   -0.4039
>>k =
      2.0000
```

将传递函数模型转化为状态空间模型：

```
>> num=[2  -1  -5];
>>den=[1  3  6  2];
>>[F,G,C,D]=tf2ss(num,den)
```

结果为

```
>>F =
    -3.0000   -6.0000   -2.0000
     1.0000         0         0
          0    1.0000         0
>>G =
      1
      0
      0
>>C =
     2.0000   -1.0000   -5.0000
>>D =
     0
```

第 5 章将对系统仿真作详细的介绍，在此不再赘述。

3.3　连续系统模型及表示

3.3.1　连续系统的基本概念

与离散系统不同，连续系统是指系统输出在时间上连续变化，而非仅在离散的时刻采样取值。连续系统的应用非常广泛，下面给出连续系统的基本概念。

【定义 3.4】　**连续系统**。满足如下条件的系统为连续系统：

(1) 系统输出连续变化。变化的间隔为无穷小量。

(2) 对系统的数学描述来说，存在系统输入或输出的微分项（导数项）。

(3) 系统具有连续的状态。在离散系统中，系统的状态为时间的离散函数，而连续系统的状态为时间连续量。

3.3.2 连续系统的数学描述

设连续系统的输入变量为 $u(t)$，其中 t 为连续取值的时间变量，设系统的输出为 $y(t)$；由连续系统的基本概念可以写出连续系统的最一般的数学描述，即

$$y(t)=f_c(u(t),t)$$

系统的实质为输入变量到输出变量的变换，注意这里系统的输入变量与输出变量既可以是标量（单输入单输出系统），也可以是向量（多输入多输出系统）；而且在系统的数学描述中含有系统输入或输出的导数。

除了采用最一般的数学方程描述连续系统外，还可以使用连续系统的微分方程形式对连续系统进行描述，即

$$\dot{x}(t)=f_c(x(t),u(t),t)\rightarrow 微分方程$$
$$y(t)=g(x(t),u(t),t)\rightarrow 输出方程$$

这里 $x(t)$、$\dot{x}(t)$ 分别为连续系统的状态变量、状态变量的微分。对于线性连续系统来说，由连续系统的微分方程描述可以容易地推导出连续系统的状态空间模型。这与使用差分方程对离散系统进行描述相类似。下面举例说明连续系统的数学描述。

【例 3.6】 对于如下的连续系统：

$$y(t)=u(t)+\dot{u}(t)，其中 u(t)=t+\sin t, t\geqslant 0$$

显然此系统为单输入单输出连续系统，且含有输入变量的微分项。由此方程可以很容易得出系统的输出变量为

$$y(t)=t+\sin t+1+\cos t=t+\sin t+\cos t+1, t\geqslant 0$$

3.3.3 连续系统的 Simulink 描述

前面给出了连续系统的基本概念与系统的基本描述方法：数学方程描述与微分方程描述。本部分使用【例 3.6】给出的连续系统

$$y(t)=u(t)+\dot{u}(t)，其中 u(t)=t+\sin t, t\geqslant 0$$

说明如何利用 Simulink 对连续系统进行描述，并在此基础上对连续系统进行简单分析。与前面类似，在此并不建立系统的 Simulink 模型进行仿真，而是采用编写 M 脚本文件从原理上对连续系统进行说明，并进行简单的仿真。

【例 3.7】 编写脚本文件 systemdemo3.m，对【例 3.6】中的连续系统进行分析。

```
% systemdemo3.m 脚本文件
t=0:0.1:5;                    %系统仿真范围，时间间隔为 0.1 s
ut=t+sin(t);                  %系统输入变量 u(t)
utdot=1+cos(t);               %系统输入变量的导数
yt=ut+utdot;                  %系统输出
plot(yt);grid;                %绘制系统输出曲线
```

图 3.4 为此连续系统在时间[0, 5]内的输出曲线。由此可见，使用简单的 MATLAB 语句可对系统性能进行简单的分析。

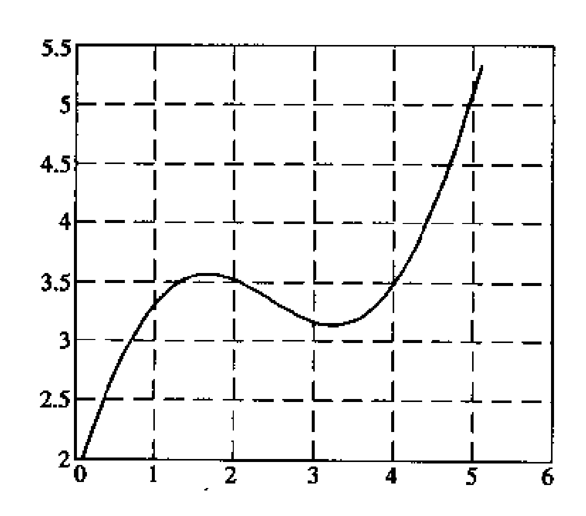


图 3.4　连续系统输入输出关系图

3.3.4　线性连续系统

在介绍线性离散系统时，已经给出线性系统的基本概念，这里做一个简单的回顾并介绍线性连续系统的概念。连续系统可以用如下的方式来表达：

$$y(t)=T\{u(t)\}$$

【定义 3.5】　**线性连续系统**。如果一个连续系统能够同时满足如下的性质：

(1) 齐次性。对于任意的参数α，系统满足

$$T\{\alpha u(t)\}=\alpha T\{u(t)\}$$

(2) 叠加性。对于任意输入变量$u_1(t)$与$u_2(t)$，系统满足

$$T\{u_1(t)+u_2(t)\}=T\{u_1(t)\}+T\{u_2(t)\}$$

则此连续系统为线性连续系统。

下面举例说明。如对【例 3.6】中的连续系统：

$$y(t)=u(t)+\dot{u}(t)，其中 u(t)=t+\sin t, t\geqslant 0$$

由于$T\{\alpha u(t)\}=\alpha t+\sin\alpha t+1+\cos\alpha t\neq\alpha T\{u(t)\}=\alpha t+\alpha\sin t+1+\alpha\cos t$，所以系统不满足齐次性，故此系统不是线性连续系统。

3.3.5　线性连续系统的数学描述

线性连续系统最一般的描述为连续系统的输入输出方程形式，即 $y(t)=f_c(u(t),t)$，也可以使用连续系统的微分方程模型进行描述：

$$\dot{x}(t)=f_c(x(t),u(t),t)\rightarrow \text{微分方程}$$
$$y(t)=g(x(t),u(t),t)\rightarrow \text{输出方程}$$

除了使用这两种连续系统通用的形式描述线性连续系统之外，还可以使用传递函数、零极点模型与状态空间模型对其进行描述。与线性离散系统相类似，线性连续系统的传递函数模型与零极点模型采用连续信号的拉氏变换来实现。

这里首先给出拉氏变换的基本定义。对于连续信号 $u(t)$，其拉氏变换定义为 $U(s)=\int_{-\infty}^{\infty}u(t)e^{-st}dt$。一般而言，系统的输入时间 $t\geqslant 0$，这时 $U(s)=\int_{0}^{\infty}u(t)e^{-st}dt$。一般将其简记为 $U(s)=L(u(t))$。拉氏变换具有如下两个性质：

(1) 线性性。即对于连续信号 $u_1(t)$ 和 $u_2(t)$，设它们的拉氏变换分别为 $U_1(s)$ 与 $U_2(s)$，则拉氏变换的线性性是指拉氏变换满足下面的关系：

$$L\{\alpha u_1(t)+\beta u_2(t)\}=\alpha L\{u_1(t)\}+\beta L\{u_2(t)\}$$

(2) 设连续信号 $u(t)$ 的拉氏变换为 $U(s)$，则 $\dot{u}(t)$ 的拉氏变换为 $sU(s)$，$\ddot{u}(t)$ 的拉氏变换为 $s^2U(s)$。

下面举例说明如何利用拉氏变换对线性连续系统进行描述。对于如下的线性连续系统：

$$u(t)=m\ddot{y}(t)+b\dot{y}(t)+ky(t)\text{，其中 } m\neq 0$$

同时对等式的两边进行拉氏变换，则有 $U(s)=ms^2Y(s)+bsY(s)+kY(s)$。将其化为分式的形式，则有

$$\frac{Y(s)}{U(s)}=\frac{1}{ms^2+bs+k}$$

这便是系统的传递函数模型。

一般来说，线性连续系统的拉氏变换总可以写成如下传递函数的形式：

$$\frac{Y(s)}{U(s)}=\frac{n_0s+n_1}{d_0s^2+d_1s+d_2}$$

将其进行一定的等价变换，可以得出线性连续系统的零极点模型：

$$\frac{Y(s)}{U(s)}=k\frac{s-z_1}{(s-p_1)(s-p_2)}$$

其中 z_1 为线性连续系统的零点，p_1、p_2 为系统的极点，k 为系统的增益。

线性连续系统的另外一种模型为状态空间模型。前面已经提到，对于线性连续系统，使用其微分方程很容易推导出系统的状态空间模型。这里给出线性连续系统用状态空间模型进行描述的一般方式：

$$\begin{cases}\dot{x}(t)=Ax(t)+Bu(t)\\ y(t)=Cx(t)+Du(t)\end{cases}$$

其中，$x(t)$ 为线性连续系统的状态变量，$u(t)$、$y(t)$ 分别为系统的输入与输出变量，可以为标量，也可以为向量。下面介绍如何在 Simulink 中实现对线性连续系统的描述。

3.3.6　线性连续系统的 Simulink 描述

一般来说，在 Simulink 中对线性连续系统的描述方式有以下三种：

(1) 线性连续系统的传递函数模型描述：在 Simulink 中，传递函数表示为 num=[n0,n1]; den=[d0,d1,d2]; 其中 num 表示传递函数的分子系数向量，den 为分母系数向量。

(2) 线性连续系统的零极点模型描述：在 Simulink 中，零极点模型表示为 gain=k; zeros=z1; poles=[p1,p2]; 其中 gain 表示系统增益，zeros 表示系统零点，poles 表示系统极点。

(3) 线性连续系统的状态空间模型描述：如果系统的状态空间表示为

$$\begin{cases} \dot{x}(t) = \boldsymbol{A}x(t) + \boldsymbol{B}u(t) \\ y(t) = \boldsymbol{C}x(t) + \boldsymbol{D}u(t) \end{cases}$$

则在 Simulink 中直接输入变换矩阵 $\boldsymbol{A}$，$\boldsymbol{B}$，$\boldsymbol{C}$，$\boldsymbol{D}$ 即可。

一般来说，线性连续系统的不同模型之间可以相互转化，MATLAB 中有内置的函数可以完成线性连续系统模型间的转化。我们在线性离散系统模型间转化中已经做了介绍，这里仅列出这些函数原型：

```
[zeros,poles,k]=tf2zp(num,den);
[num,den]=zp2tf(zeros,poles,k);
[zeros,poles,k]=ss2zp(A,B,C,D);
[A,B,C,D]=zp2ss(zeros,poles,k)
[num,den]=ss2tfA,B,C,D)
[A,B,C,D]=tf2ss(num,den)
```

在知道线性连续系统的描述模型之后，可以很方便地对系统进行仿真分析，下面举例说明。

【例 3.8】 对于如下采用传递函数模型进行描述的线性连续系统：

$$\frac{Y(s)}{U(s)} = \frac{s-3}{2s^2-3s-5}$$

要求绘制此系统的 Bode 图、Nyquist 图，并求取系统的零极点模型与状态空间模型描述。

解：在 MATLAB 中输入下面的语句即可：

```
>>num=[1, -3];
>>den=[2, -3, -5];
>>w=logspace(-1, 1);
>>subplot(2,1,1); bode(num, den, w);
>>subplot(2,1,2); nyquist(num,den,w);
>>[zeros, poles, k]=tf2zp(num,den)
>>[A,B,C,D]=tf2ss(num,den)
```

系统的 Bode 图与 Nyquist 图如图 3.5 所示。

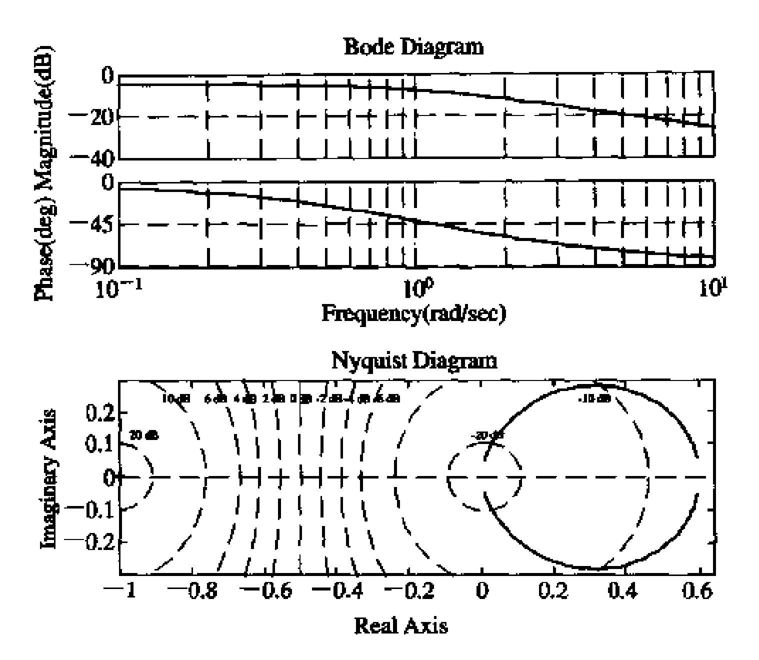


图 3.5 线性连续系统的 Bode 图与 Nyquist 图

系统的零极点模型与状态空间模型如下所示：

```
>>zeros =
     3
poles =
    2.5000
   -1.0000
k =
    0.5000
A =
    1.5000    2.5000
    1.0000    0
B =
     1
     0
C =
    0.5000   -1.5000
D =
     0
```

3.4 混合系统模型及表示

本章前三节分别对简单系统、离散系统以及连续系统做了简单的介绍。对于单独类型的系统，对其进行分析仿真比较容易。但是在实际的系统中，系统往往非常复杂，并且往

往由不同类型的系统共同构成。这样的系统称之为混合系统（或混杂系统），它是由离散系统与连续系统等共同构成的系统。虽然混合系统一般都比较复杂，但是只要将系统进行合理的划分，将系统分解为不同的部分，分别对每一部分进行分析，最后再对整个系统进行综合分析，则会大大减小系统仿真分析的复杂性。

3.4.1　混合系统的数学描述

混合系统是由不同类型的系统共同构成的，因此混合系统的数学描述可以由不同类型系统描述共同构成。但是由于混合系统的复杂性，一般难以用单独的数学模型进行描述或表达，因此混合系统一般都是由系统各部分输入与输出间的数学方程所共同描述的，下面举例说明。

【例 3.9】　对于如下的一个混合系统：设系统的输入为一离散变量 $u(n), n=1,2,3\cdots\cdots$，系统由离散系统与连续系统串联构成，其中离散系统输出经过一个零阶保持器后作为连续系统的输入。其中离散系统的输入输出方程为 $y(n)=u(n)+1$ 且 $u(n)=n/2$，系统采样时间为 T_s=1 s。

连续系统的输入输出方程为

$$y(t)=\sqrt{u(t)}+\sin u(t)$$

由于此混合系统中离散系统的输出 $y(n)$ 经过一零阶保持器后作为连续系统的输入，因此 $u(t)$ 与 $y(n)$ 的数学关系为

$$u(t)=y(n),\quad nT_s\leqslant t<(n+1)T_s$$

其中 T_s=1 s 为离散系统的采样时间。故此混合系统的输入与输出之间的关系可以由下面的方程来描述：

$$\begin{cases}u(n)=n/2, & n=1,2,3\cdots\cdots\\ y(n)=u(n)+1, & \\ y(t)=\sqrt{y(n)}+\sin(y(n)), & n\leqslant t<n+1\end{cases}$$

由此可见，对于混合系统而言，离散系统的采样时间是一个非常重要的系统参数，在单独的离散系统中，一般只需关心离散系统在固定的离散时刻的输出，而并不需要指出具体的时刻，这一点与混合系统不相同。

3.4.2　混合系统的 Simulink 描述与简单分析

在对单独离散系统或连续系统进行描述时，由于系统一般比较简单，因而可以采用诸如差分方程、传递函数、状态空间等模型表示。但对于混合系统，由于系统本身的复杂性，即使是很简单的混合系统，如【例 3.9】给出的例子，都难以用一个简单的模型进行描述。因此，这里采用简单的数学方式对系统进行描述与分析。

【例 3.10】　编写 M 脚本文件 systemdemo4.m，对【例 3.9】中的混合系统进行分析。

```
%systemdemo4.m 文件
t=1:0.1:99.9;              %表示在时间[1, 99.9]范围内分析系统。时间间隔 0.1 s
n=1:100;                   %表示系统输入时刻为 1~100 s
```

```
un=0.5*n;                %表示系统输入 u(n)
yn=un+1;                 %表示系统中离散部分的输出，即连续部分的输入
for i=1:length(n)-1
   for j=1:length(t)
      if t(j)>=n(i)&t(j)<n(i+1)            %判断连续部分的输入时间范围
         y(j)=sqrt(yn(n(i)))+sin(yn(n(i)));   %计算系统输出
      end
   end
end
plot(t,y);grid;          %绘制系统输出曲线图
```

系统输出曲线图如图 3.6 所示。

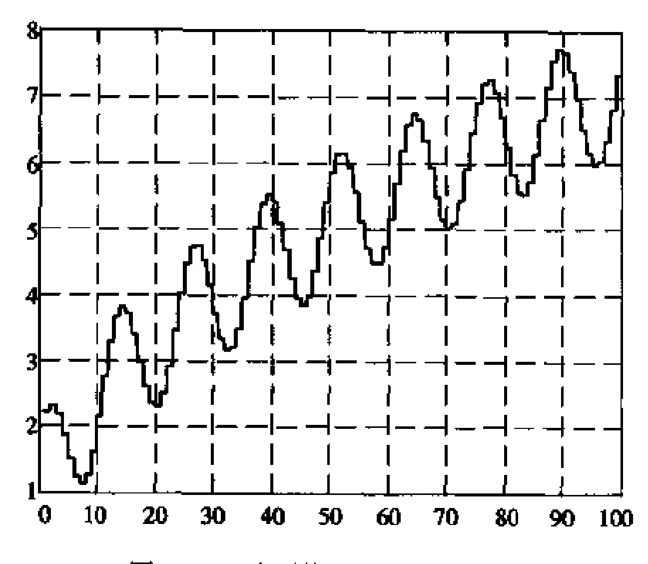
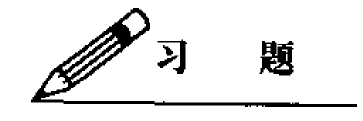


图 3.6 混合系统输入输出关系图

从系统输出曲线图 3.6 中可以看出：由于系统中离散部分的输出经过零阶保持器后作为连续部分的输入，而零阶保持器具有阶跃的特性，在系统仿真结果中出现阶跃现象。另外，系统呈现类似正弦发散的特征表明系统为一发散不稳定系统。

本章对动态系统做了简单的介绍，其中涉及到简单系统、离散系统、线性离散系统、连续系统、线性连续系统、混合系统等系统的概念以及数学描述和 Simulink 描述，并且使用简单的 MATLAB 语句对不同的系统进行了简单的仿真与分析。至于使用 Simulink 进行动态系统建模、仿真与分析将在第二部分进行详细介绍。



习 题

1. 求简单系统 $y(x)=\begin{cases}-x+5, & x<10\\(1+x)\sin x, & x\geqslant 10\end{cases}$ 的零点。

提示：系统零点相当于函数的根，使用 roots 命令分段求解。

2. 使用滤波器形式表示线性离散系统：$Y(z)=U(z)+z^{-1}U(z)+2z^{-1}Y(z)$，并绘制其 Bode 图。

3. 使用不同的模型在 MATLAB 中描述线性连续系统：$\frac{Y(s)}{U(s)}=\frac{s-1}{s^2+3s-4}$。



第二部分

进阶

Simulink 动态系统仿真

第二部分的内容是全书的核心。在第一部分所介绍的动态系统仿真技术的基础之上，本部分将详细介绍如何使用 Simulink 建立动态系统模型、对动态系统进行仿真分析，进而对动态系统进行设计分析。本部分内容的组织与讲解，适于各个领域系统设计者的学习与理解。本部分包括如下内容：

➤创建 Simulink 模型：主要介绍动态系统模型建立、模型界面设计以及 Simulink 与 MATLAB 之间的接口设计。

➤动态系统的 Simulink 仿真：主要介绍各种类型动态系统的仿真技术、综合系统设计分析与系统调试技术。

第4章 创建 Simulink 模型

内容概要

· 启动 Simulink 并建立系统模型
· Simulink 模块库简介与使用
· 构建 Simulink 系统框图
· 设计 Simulink 系统框图界面
· Simulink 与 MATLAB 的接口
· 使用 Simulink 进行简单的仿真

第一部分对使用 Simulink 应该具备的一些基础知识做了简单的介绍。虽然用户能够直接利用 MATLAB 中的基本语句与函数对动态系统进行分析，但由于当前各应用领域中的系统设计与分析变得越来越复杂，直接使用 MATLAB 对复杂系统进行分析费时费力，难以有效地实现系统仿真分析的要求。

Simulink 是基于 MATLAB 的图形化仿真环境。它使用图形化的系统模块对动态系统进行描述，并在此基础上进行动态系统的求解。Simulink 采用 MATLAB 的计算引擎对动态系统在时域内进行求解，此计算引擎的主要功能如下：

(1) 计算各动态系统中各模块的输出。

(2) 传递信号到下一个系统模块。

利用 Simulink 对动态系统进行仿真的核心在于，MATLAB 计算引擎对系统微分方程和差分方程的求解。Simulink 与 MATLAB 是高度集成在一起的，因此 Simulink 和 MATLAB 之间可以进行灵活的交互操作。此外，Simulink 作为 MathWorks 公司推出的系统级的仿真平台，使用高级图形技术，具有优秀的用户界面，并且提供大量的系统内置模块。图形化的仿真与设计环境使得 Simulink 的应用越来越广泛。本章将主要介绍如何使用 Simulink 图形仿真环境建立动态系统的 Simulink 模型。

4.1 启用 Simulink 并建立系统模型

由于 Simulink 是基于 MATLAB 环境之上的高性能的系统级仿真设计平台，因此启动 Simulink 之前必须首先运行 MATLAB，然后才能启动 Simulink 并建立系统模型。启动 Simulink 有两种方式：

(1) 用命令行方式启动 Simulink。即在 MATLAB 的命令窗口中直接键入如下命令：

```
>>simulink
```

(2) 使用工具栏按钮启动 Simulink。即用鼠标单击 MATLAB 工具栏中的 Simulink 按钮。启动 Simulink，建立系统模型，其相应的基本操作如图 4.1 所示。

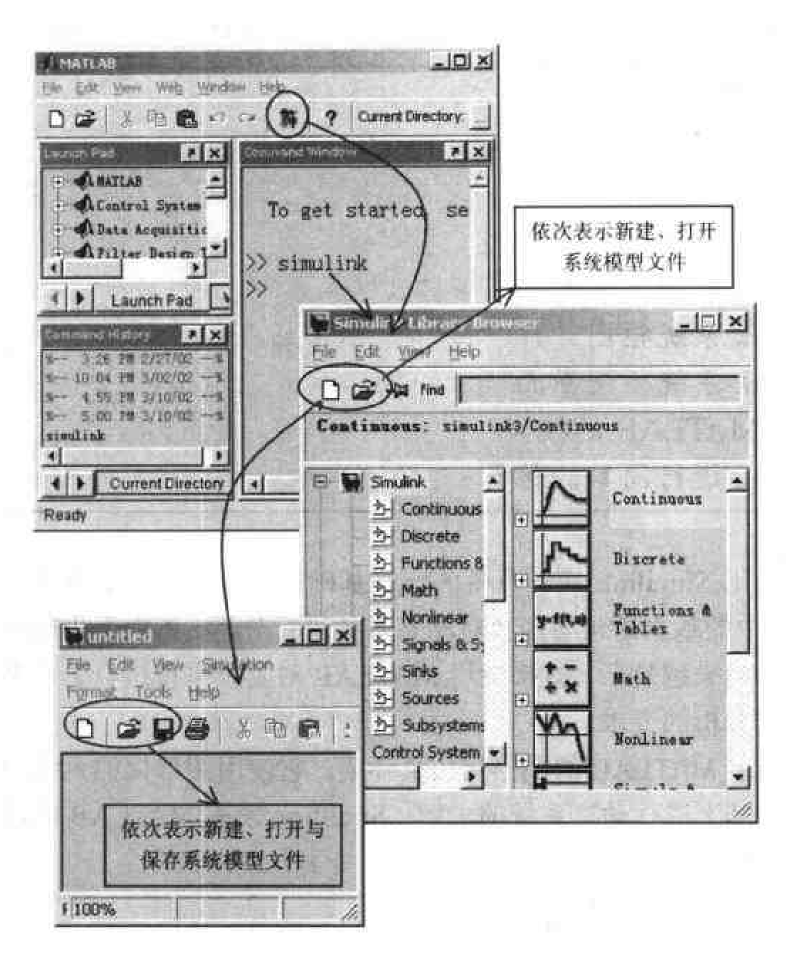


图 4.1 启动 Simulink，建立系统模型的基本操作

说明：图 4.1 中从上到下，依次为 MATLAB 主窗口、Simulink 主窗口（即 Simulink 库浏览器窗口）以及系统模型编辑器，图中的箭头反映了操作的顺序。当打开一个新的系统模型文件后，用户便可以从 Simulink 模块库中选择合适的系统模块或者使用自定义的模块来建立系统模型。建立系统模型有如下的几种方式：

(1) 单击 Simulink 主窗口中 File 菜单下的 New Model 子菜单，创建新的系统模型。

(2) 单击 Simulink 主窗口中的 Create a new model 按钮。

(3) 单击 MATLAB 主窗口中 File 菜单下的 New Model 子菜单。

当用户完成 Simulink 系统模型的编辑之后，需要保存系统模型，然后设置模块参数与系统仿真参数，最后便可以进行系统的仿真。

无论采用何种方式，用户都可以在短短几分钟内熟练掌握启动 Simulink 的方法并开始创建动态系统模型。在系统模型编辑器中，用户可以“拖动”Simulink 提供的大量的内置模块建立系统模型。下一节将对 Simulink 中的内置系统模块作一个比较全面的介绍，以便初学者无需查阅各个模块的帮助文献，便可以迅速建立所需的系统模型。

4.2　Simulink 模块库简介与使用

在 4.1 节中，用户已经掌握了如何启动 Simulink 并新建一个动态系统模型。为便于用户能够快速构建自己所需的动态系统，Simulink 提供了大量以图形方式给出的内置系统模块，使用这些内置模块可以快速方便地设计出特定的动态系统。为了便于用户对 Simulink 内置模块库的认识与使用，本节简单介绍 Simulink 中的模块库以及模块库中具有代表意义的系统模块。图 4.2 所示为 Simulink 的模块库浏览器。

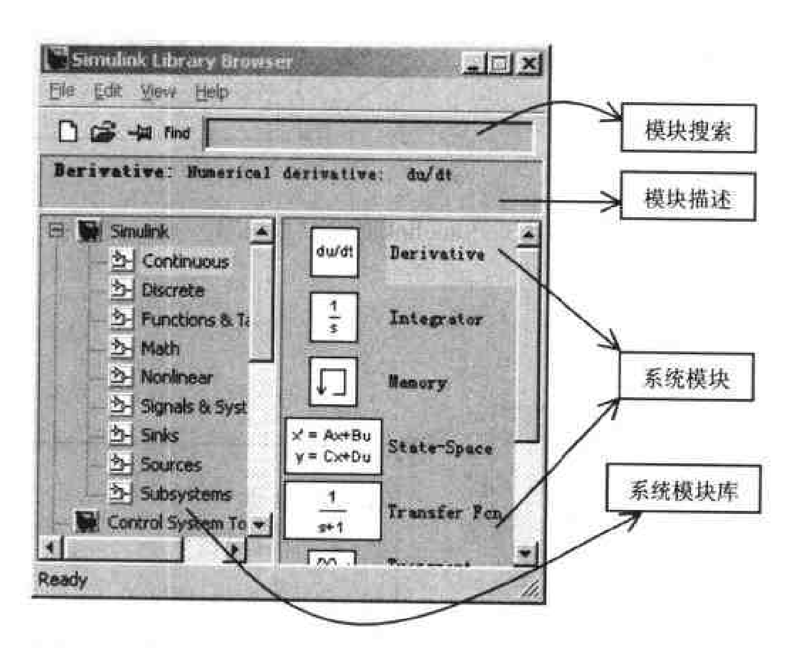


图 4.2　Simulink 的模块库浏览器

Simulink 的模块库能够对系统模块进行有效的管理与组织，使用 Simulink 模块库浏览器可以按照类型选择合适的系统模块、获得系统模块的简单描述以及查找系统模块等，并且可以直接将模块库中的模块拖动或者拷贝到用户的系统模型中以构建动态系统模型。下面仅对最为常用的模块库以及库中的常用模块进行简单的介绍，以使用户对其有个大概的了解，并能够使用这些模块建立自己的动态系统模型。

4.2.1　Simulink 公共模块库

Simulink 公共模块库是 Simulink 中最为基础、最为通用的模块库，它可以被应用到不同的专业领域中。Simulink 公共模块库共包含 9 个模块库，如图 4.3 所示。下面分别介绍各个模块的功能：

1. Continuous（连续系统模块库）

连续系统模块库以及其中各模块的功能如图 4.4 所示。

2. Discrete（离散系统模块库）

离散系统模块库以及其中各模块的功能如图 4.5 所示。

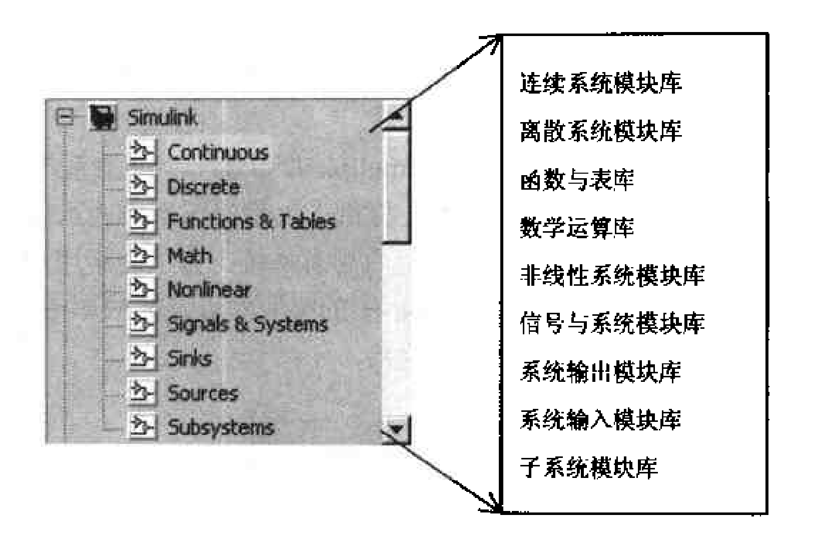


图 4.3　Simulink 的公共模块库

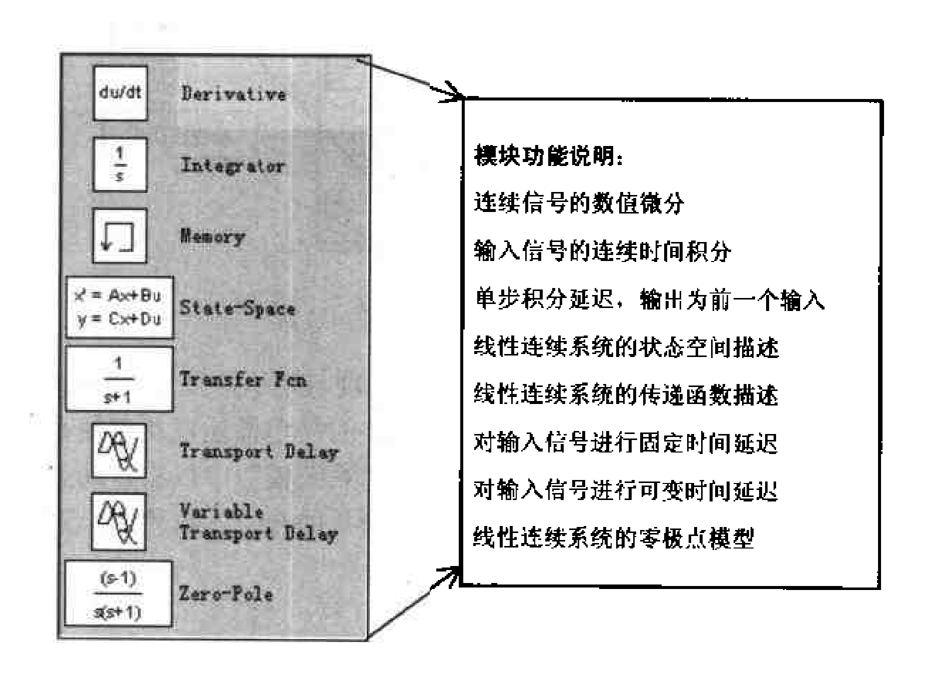


图 4.4　连续系统模块库及其功能

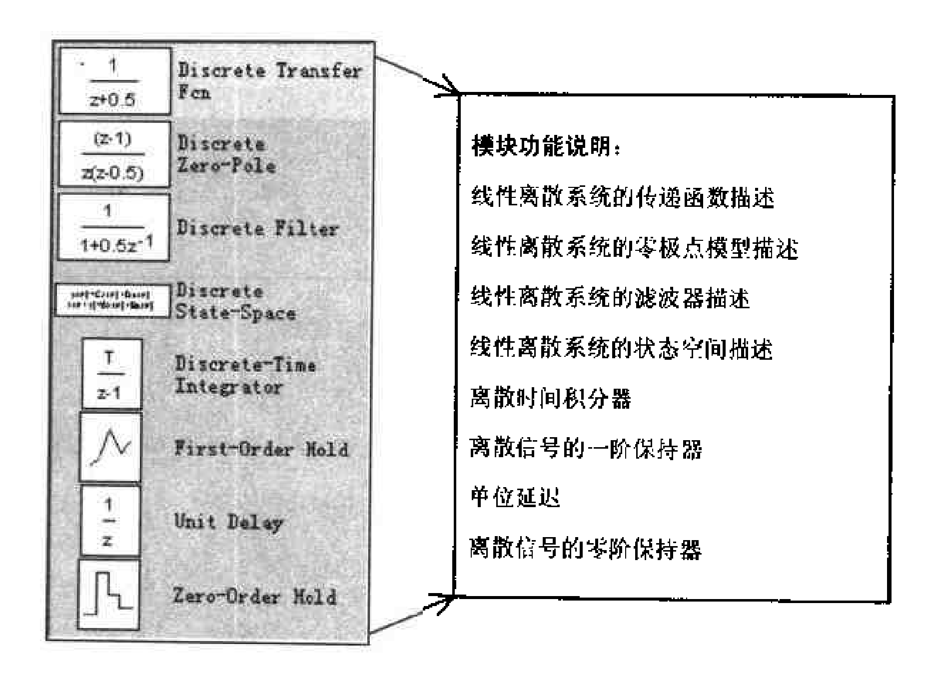


图 4.5 离散系统模块库及其功能

3. Functions & Tables（函数与表库）

函数与表库以及其中各模块的功能如图 4.6 所示。

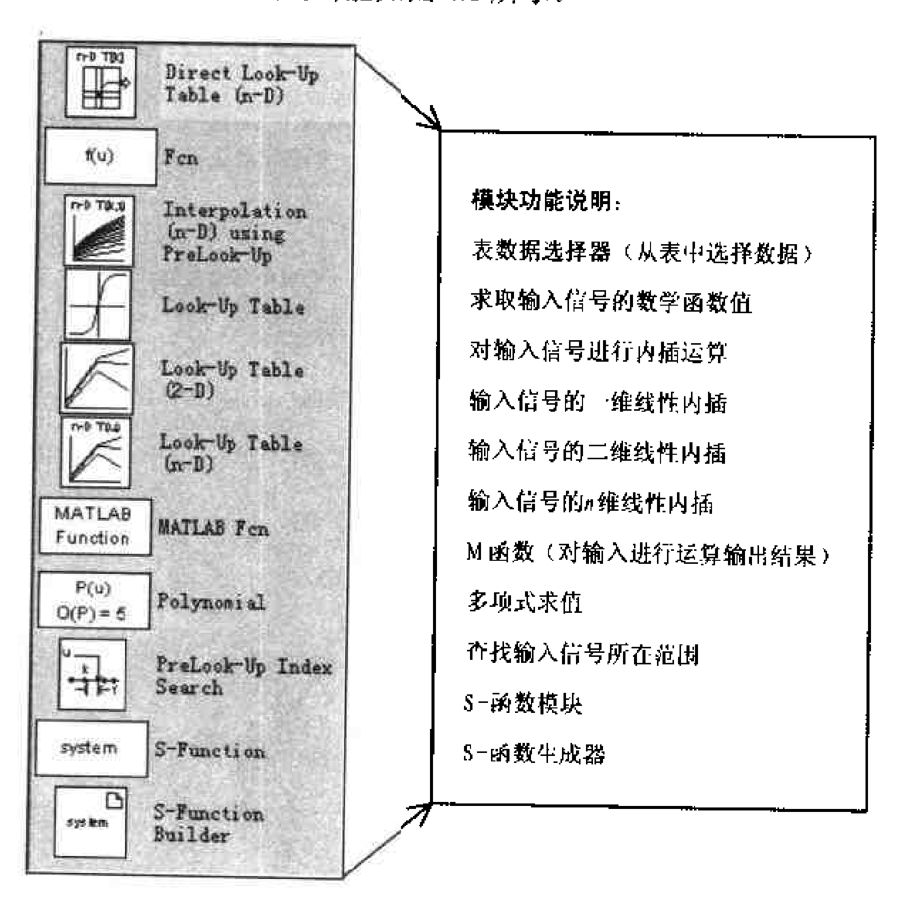


图 4.6 函数与表库及其功能

4. Math（数学运算库）

数学运算库以及其中各模块的功能如图 4.7 所示。

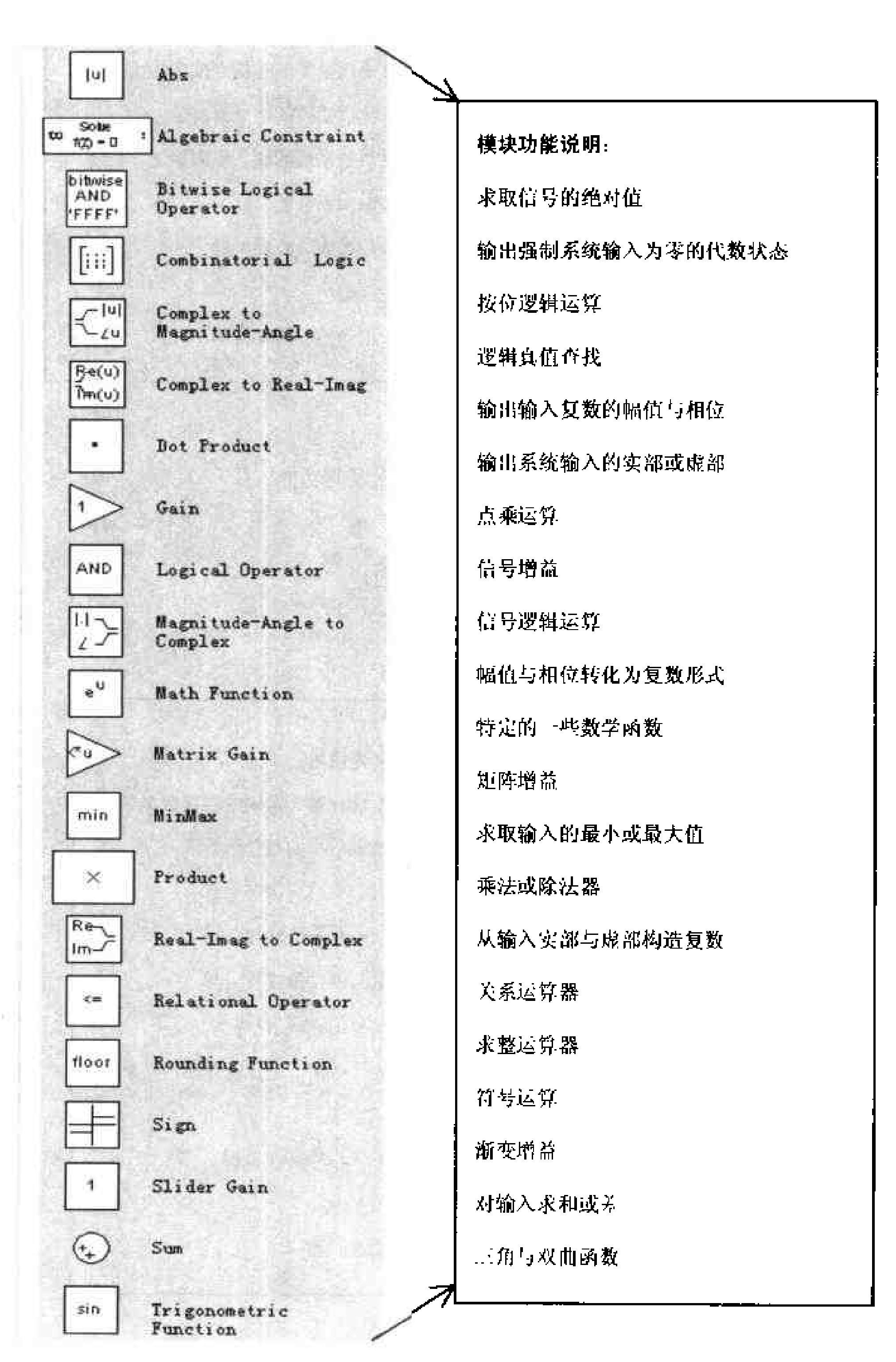


图 4.7　数学运算库及其功能

5. Nonlinear（非线性系统模块库）

非线性系统模块库以及其中各模块的功能如图 4.8 所示。

6. Signals & Systems（信号与系统模块库）

信号与系统模块库以及其中各模块的功能如图 4.9 所示。

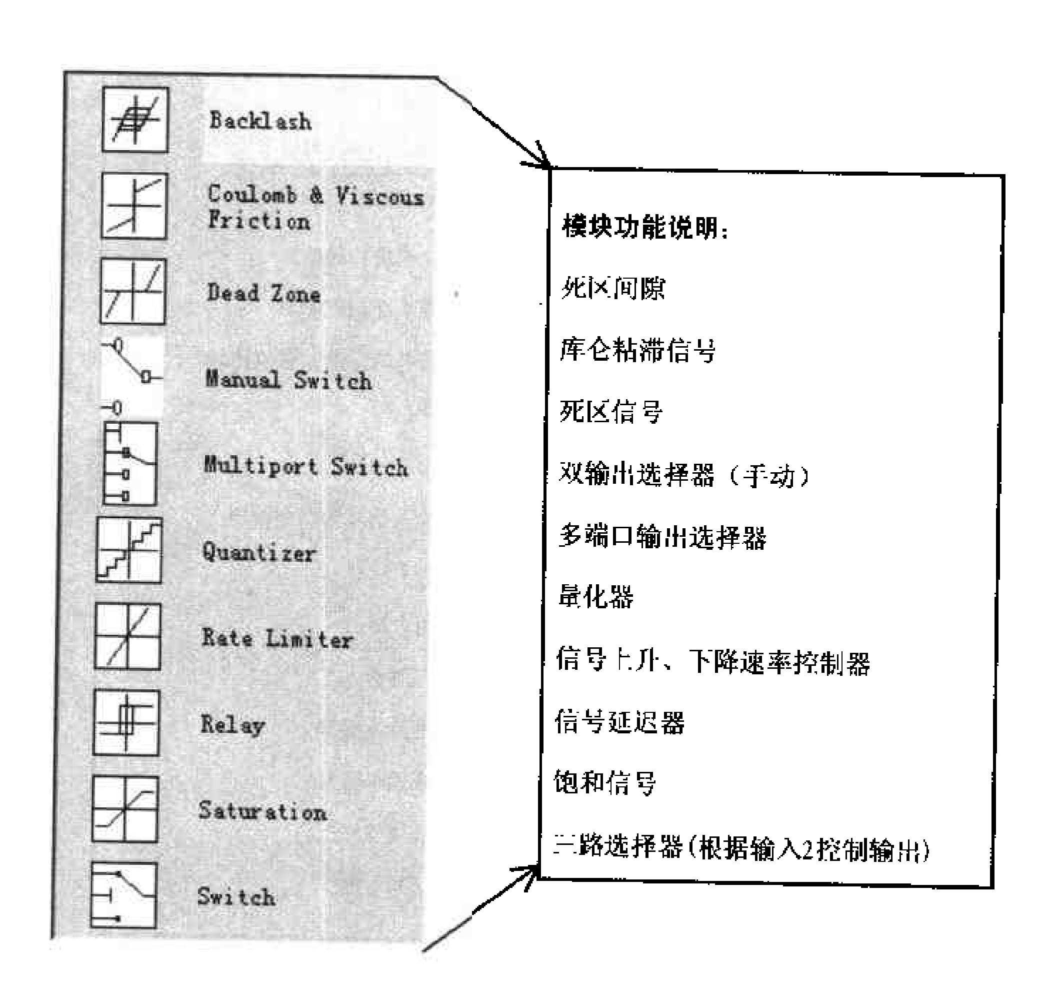
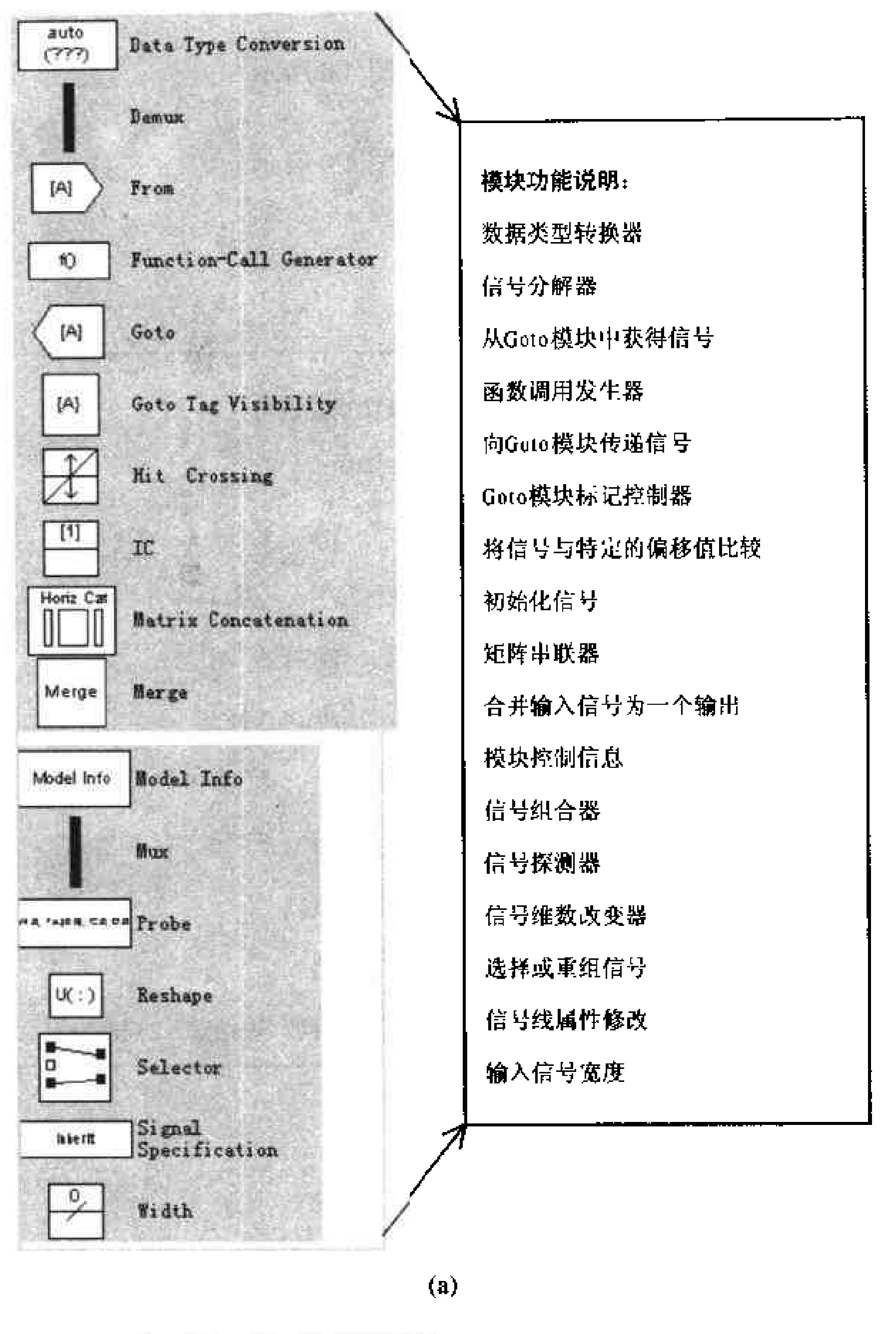


图 4.8　非线性系统模块库及其功能



(a)

Assignment
Bus Creator
Bus Selector
Data Store Memory
Data Store Read
Data Store Write

模块功能说明：
对信号进行分配
由输入产生总线信号
总线信号选择器
用户定义的数据存储区
从数据存储区中读取数据
向数据存储区写数据

(b)

图 4.9　信号与系统模块库及其功能

7. Sinks（系统输出模块库）

系统输出模块库以及其中各模块的功能如图 4.10 所示。

8. Sources（系统输入模块库）

系统输入模块库以及其中各模块的功能如图 4.11 所示。

9. Subsystems（子系统模块库）

子系统模块库以及其中各模块功能如图 4.12 所示。

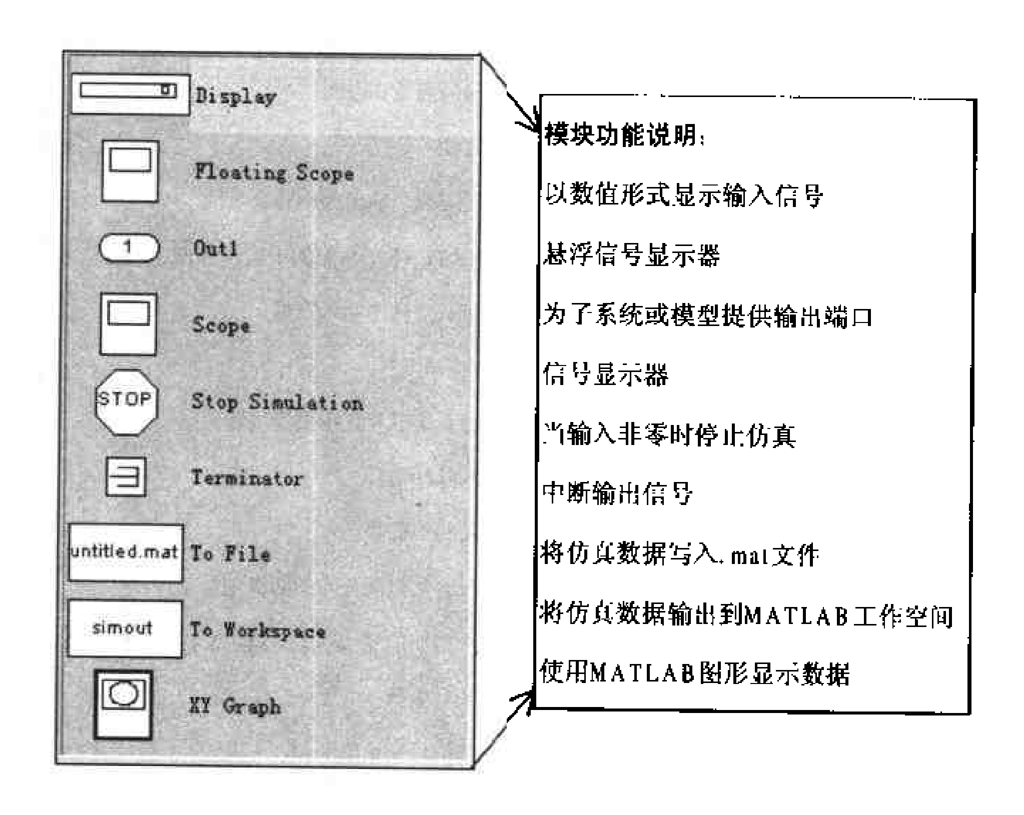


图 4.10　系统输出模块库及其功能

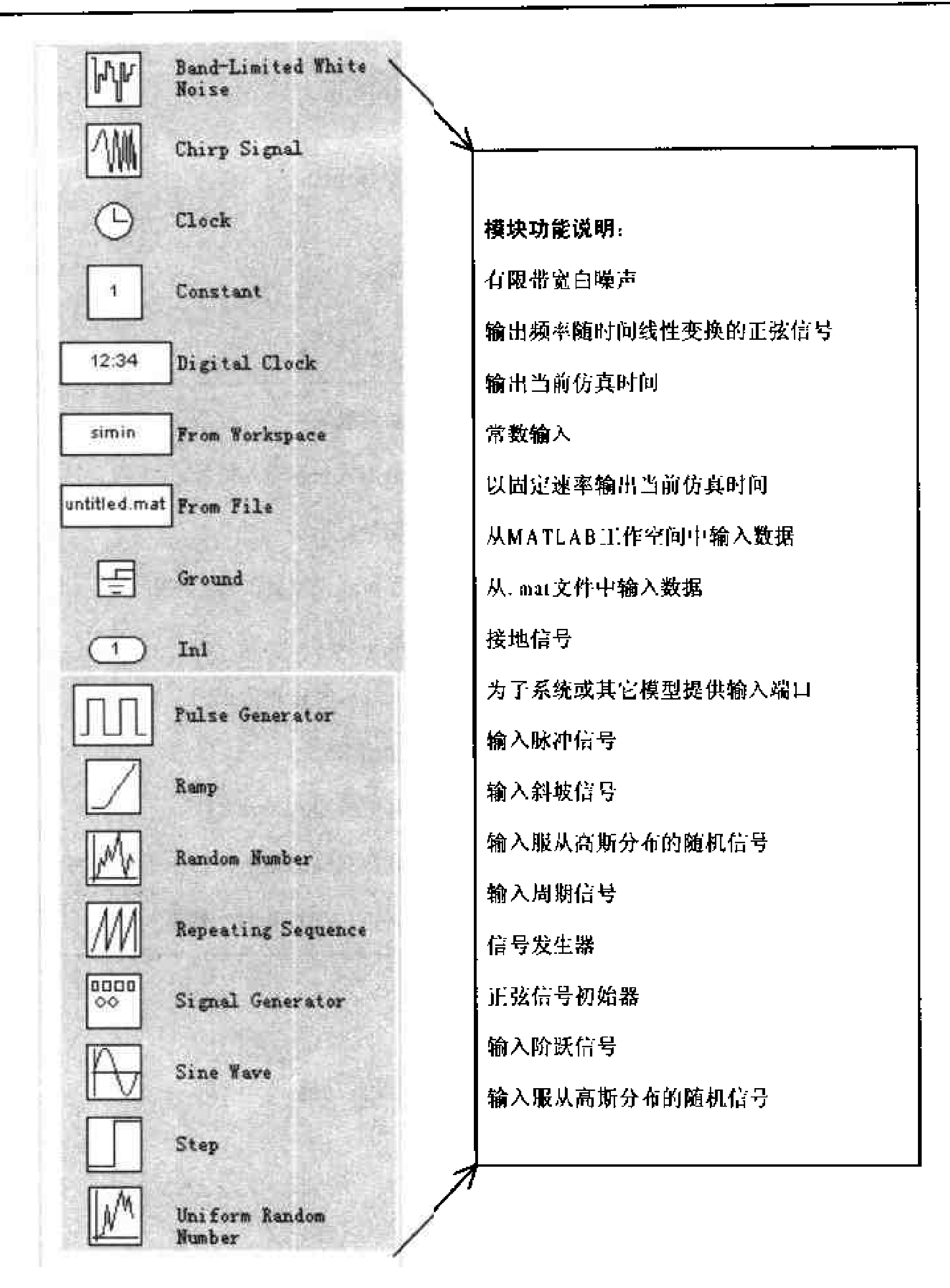


图 4.11 系统输入模块库及其功能

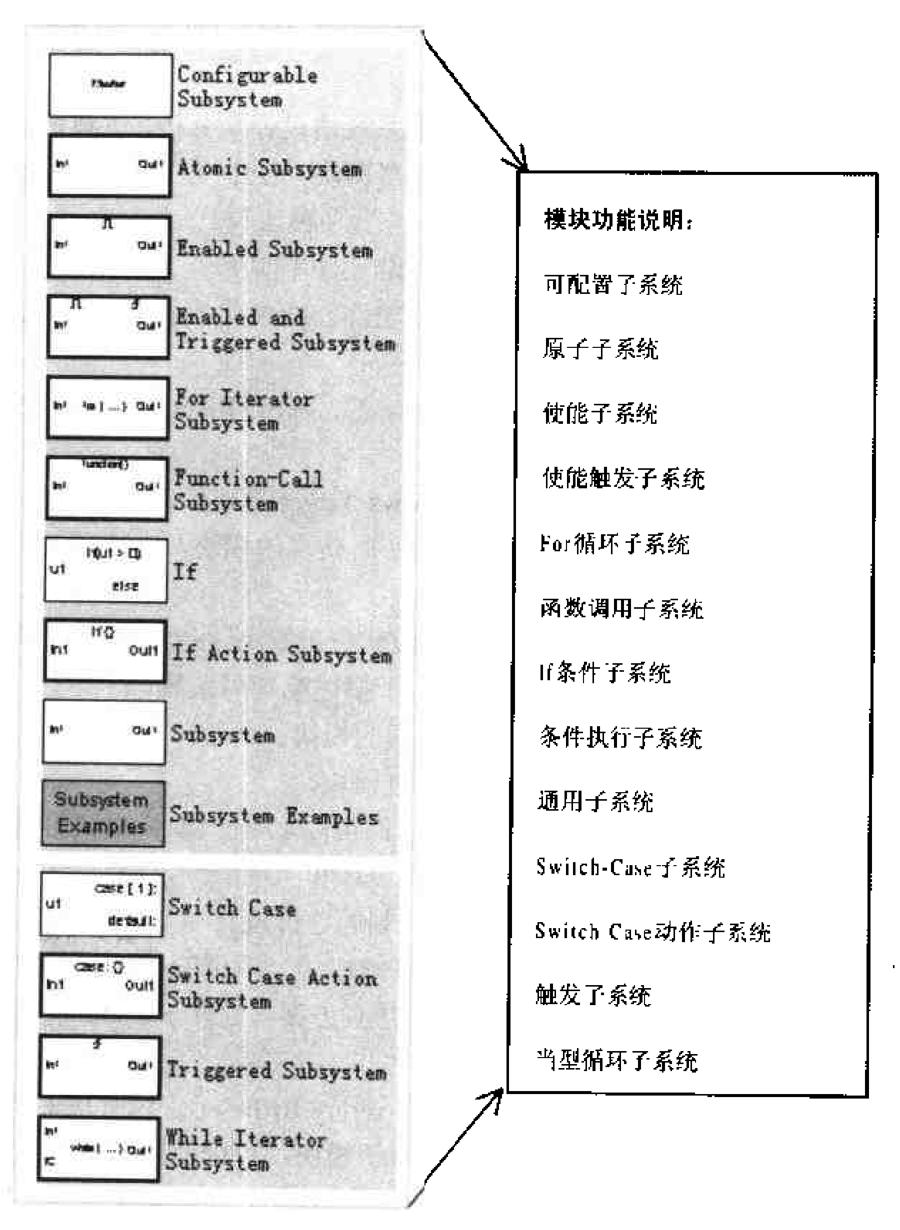


图 4.12　子系统模块库及其功能

之所以用较多的篇幅对 Simulink 的公共模块库进行比较全面的介绍，是因为 Simulink 的公共模块库中提供了大量内置的系统模块，这些系统模块的用途非常广泛，并且一般的动态系统模型都可以使用公共模块库中的模块来构建。

除了公共模块库之外，Simulink 中还集成了许多面向不同专业领域的专业模块库，普通用户一般很少用到其中的模块。因此，在介绍 Simulink 的专业模块库时，仅对模块库的总体功能做简单的概述。如果用户需要的话，可以在 Simulink 中的模块描述栏了解其主要功能。

4.2.2　Simulink 专业模块库

Simulink 集成了许多面向各专业领域的系统模块库，不同领域的系统设计者可以使用这些系统模块快速构建自己的系统模型，然后在此基础上进行系统的仿真与分析，从而完成系统设计的任务。这里仅简单介绍部分专业模块库的主要功能。

(1) Control System Toolbox 模块库：面向控制系统的设计与分析，主要提供线性时不变系统的模块。

(2) DSP Blockset 模块库：面向数字信号处理系统的设计与分析，主要提供 DSP 输入模块、DSP 输出模块、信号预测与估计模块、滤波器模块、DSP 数学函数库、量化器模块、信号管理模块、信号操作模块、统计模块以及信号变换模块等。

(3) Simulink Extras 模块库：主要补充 Simulink 公共模块库，提供附加连续模块库、附加线性系统模块库、附加输出模块库、触发器模块库、线性化模块库、系统转换模块库以及航空航天系统模块库等。

(4) S-function demos 模块库：主要提供 C++、C、FORTRAN 以及 M 文件下 S-函数的模块库的演示模块。

(5) Real-Time Workshop 与 Real-Time Windows Target 模块库：主要提供各种用来进行独立可执行代码或嵌入式代码生成，以实现高效实时仿真的模块。它们和 RTW、TLC 有着密切的联系。

(6) Stateflow 库：对使用状态图所表达的有限状态机模型进行建模仿真和代码生成。有限状态机用来描述基于事件的控制逻辑，也可用于描述响应型系统。

(7) 定点模块库：包含一组用于定点算法仿真的模块。

(8) 通讯模块库：专用于通信系统仿真的一组模块。

(9) Dials & Gauges 库：图形仪表模块库，它们实际上是一组 ActiveX 控件。

(10) 神经网络模块库：用于神经网络的分析设计和实现的一组模块。

(11) 模糊控制模块库：包括一组有关模糊控制的分析设计和实现的模块。

(12) xPC 模块库：提供了一组用于 xPC 仿真的模块。所谓的 xPC 是指利用 PC 机，使用客户端服务器的模式进行实时仿真的一种经济仿真方案。它和 Simulink、RTW 相结合，可以在 PC 机上进行单任务的实时仿真。

除了上面介绍的一些模块库之外，Simulink Library Browser 中还有很多其它的模块库。有效地利用 Simulink 系统模块库中的各种内置系统模块，可以使用户在很短的时间内完成各个领域复杂系统的建模、仿真与分析。至于如何使用这些内置的系统模块构建动态系统模型将在下一节中进行详细的介绍。

4.3 构建 Simulink 框图

4.2 节中简单介绍了 Simulink 中的一些比较常用的系统模块。本节将介绍如何使用这些系统模块以构建用户自己的系统模型。当 Simulink 库浏览器被启动之后，通过鼠标左键单击模块库的名称可以查看模块库中的模块。模块库中包含的系统模块显示在 Simulink 库浏览器右边的一栏中。对 Simulink 库浏览器的基本操作有：

(1) 使用鼠标左键单击系统模块库，如果模块库为多层结构，则单击“+”号载入库。

(2) 使用鼠标右键单击系统模块库，在单独的窗口打开库。

(3) 使用鼠标左键单击系统模块，在模块描述栏中显示此模块的描述。

(4) 使用鼠标右键单击系统模块，可以得到系统模块的帮助信息，将系统模块插入到系

统模型中，查看系统模块的参数设置，以及回到系统模块的上一层库。

此外还可以进行以下操作：

(1) 使用鼠标左键选择并拖动系统模块，并将其拷贝到系统模型中。

(2) 在模块搜索栏中搜索所需的系统模块。

4.3.1 模块选择

这里用一个非常简单的例子介绍如何建立动态系统模型。此简单系统的输入为一个正弦波信号，输出为此正弦波信号与一个常数的乘积。要求建立系统模型，并以图形方式输出系统运算结果。已知系统的数学描述为

$$\text{系统输入：}\ u(t)=\sin t\ ,t\geqslant 0$$

$$\text{系统输出：}\ y(t)=au(t),a\neq 0$$

启动 Simulink 并新建一个系统模型文件。欲建立此简单系统的模型，需要如下的系统模块（均在 Simulink 公共模块库中）：

(1) 系统输入模块库 Sources 中的 Sine Wave 模块：产生一个正弦波信号。

(2) 数学库 Math 中的 Gain 模块：将信号乘上一个常数（即信号增益）。

(3) 系统输出库 Sinks 中的 Scope 模块：图形方式显示结果。

选择相应的系统模块并将其拷贝（或拖动）到新建的系统模型中，如图 4.13 所示。

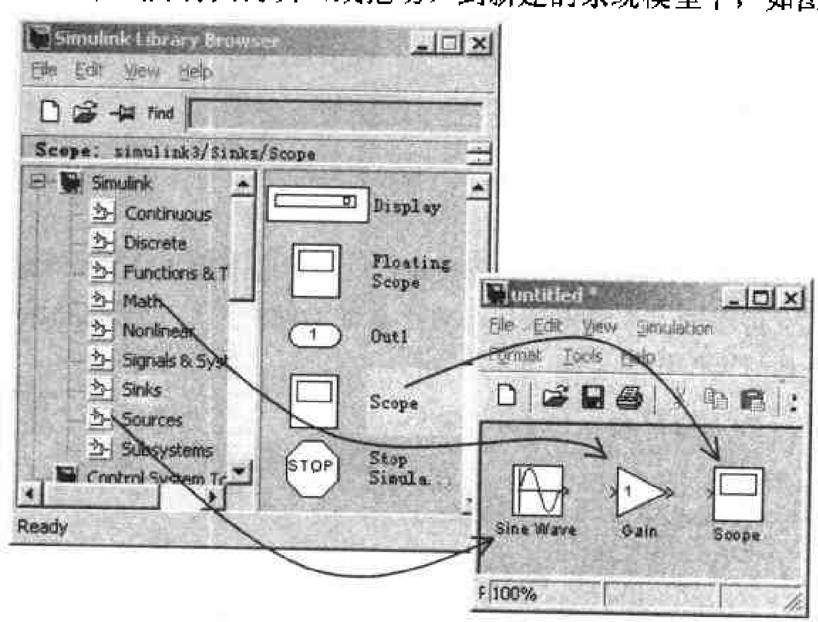


图 4.13　选择系统所需模块

在选择构建系统模型所需的所有模块后，需要按照系统的信号流程将各系统模块正确连接起来。连接系统模块的步骤如下：

(1) 将光标指向起始块的输出端口，此时光标变成“+”。

(2) 单击鼠标左键并拖动到目标模块的输入端口，在接近到一定程度时光标变成双十字。这时松开鼠标键，连接完成。完成后在连接点处出现一个箭头，表示系统中信号的流向，如图 4.14 所示。

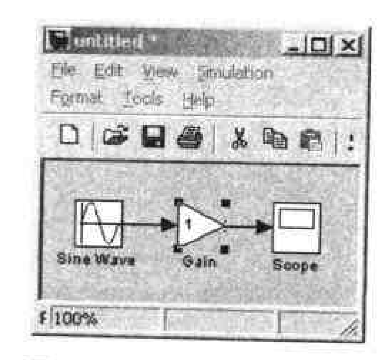


图 4.14　系统模块之间的连线

在 Simulink 的最新版本中，连接系统模块还有如下更有效的方式：

(1) 使用鼠标左键单击起始模块。

(2) 按下 Ctrl 键，并用鼠标左键单击目标块。

4.3.2 模块操作

下面介绍一些对系统模块进行操作的基本技巧，掌握它们可使建立动态系统模型变得更为方便快捷。

1. 模块的复制

如果需要几个同样的模块，可以使用鼠标右键单击并拖动某个块进行拷贝。也可以在选中所需的模块后，使用 Edit 菜单上的 Copy 和 Paste 或使用热键 Ctrl+C 和 Ctrl+V 完成同样的功能，如图 4.15 所示。

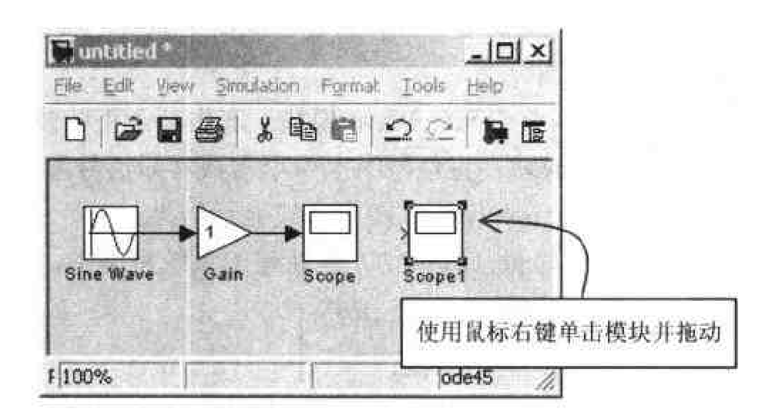


图 4.15　模块的复制

2. 模块的插入

如果用户需要在信号连线上插入一个模块，只需将这个模块移到线上就可以自动连接。注意这个功能只支持单输入单输出模块。对于其他的模块，只能先删除连线，放置块，然后再重新连线。具体操作如图 4.16 所示。

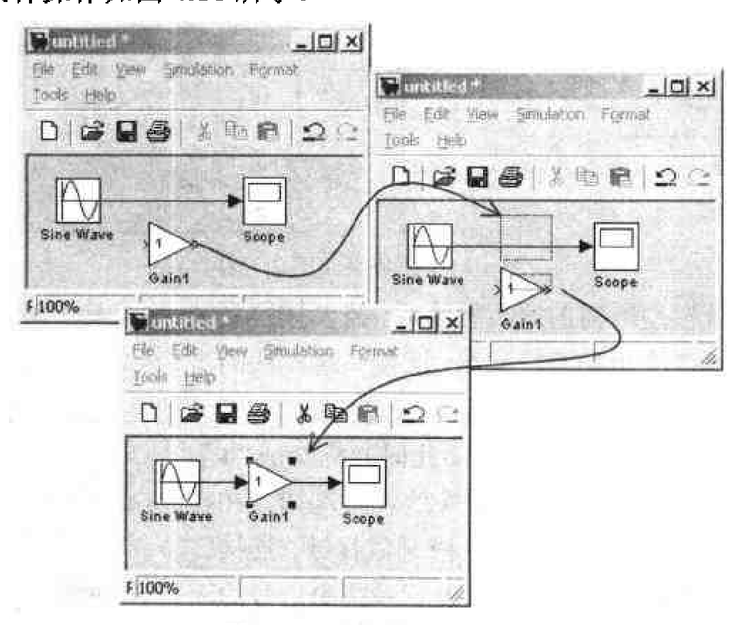


图 4.16　系统模块的插入

3. 连线分支与连线改变

在某些情况下，一个系统模块的输出同时作为多个其它模块的输入，这时需要从此模块中引出若干连线，以连接多个其它模块。对信号连线进行分支的操作方式为：使用鼠标右键单击需要分支的信号连线（光标变成“+”），然后拖动到目标模块。

对信号连线还有以下几种常用的操作：

(1) 使用鼠标左键单击并拖动以改变信号连线的路径。

(2) 按下 Shift 键的同时，在信号连线上单击鼠标左键并拖动，可以生成新的节点。

(3) 在节点上使用鼠标左键单击并拖动，可以改变信号连线路径。

信号连线分支与连线改变如图 4.17 所示。

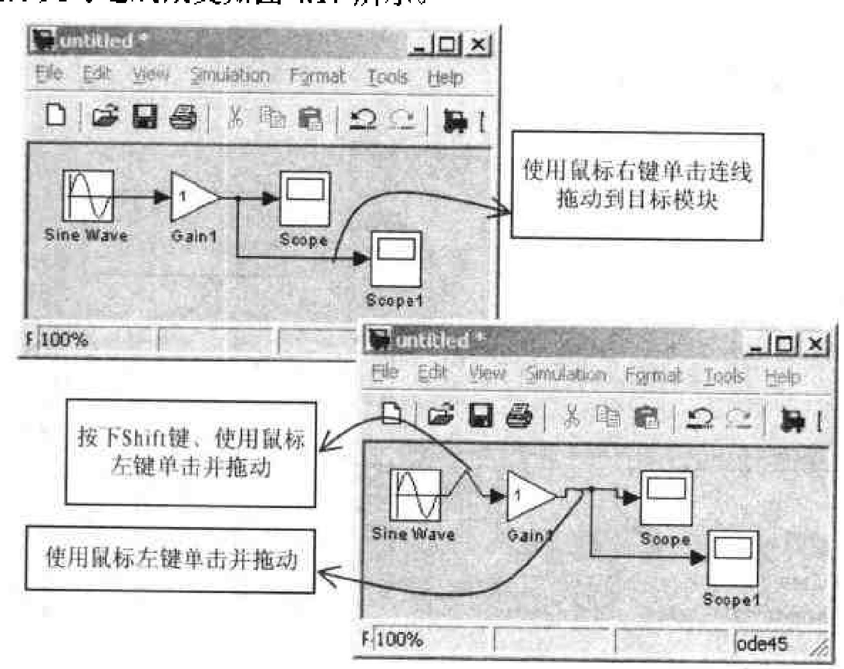


图 4.17　连线分支与连线改变

4. 信号组合

在利用 Simulink 进行系统仿真时，在很多情况下，需要将系统中某些模块的输出信号（一般为标量）组合成一个向量信号，并将得到的信号作为另外一个模块的输入。例如，使用示波器显示模块 Scope 显示信号时，Scope 模块只有一个输入端口；若输入是向量信号，则 Scope 模块以不同的颜色显示每个信号。能够完成信号组合的系统模块为 Signals and Systems 模块库中的 Mux 模块，使用 Mux 模块可以将多个标量信号组合成一个向量。因此使用 Simulink 可以完成矩阵与向量的传递。信号组合如图 4.18 所示。

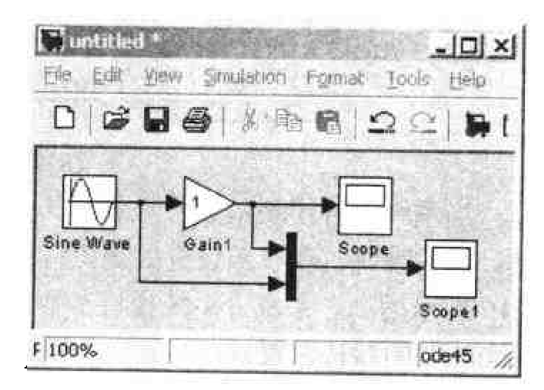


图 4.18　信号组合

如果系统模型中包含向量信号，使用 Format 菜单中的 Wide Nonscalar Lines 可以将它们区分出来（标量信号的连线较细，而向量信号的连线较粗）；也可以使用 Format 菜单中的 Signal Dimensions 显示信号的维数（在相应的信号连线上显示信号的维数），如图 4.19 所示。

4.3.3 运行仿真

1. 系统模块参数设置与系统仿真参数设置

当用户按照信号的输入输出关系连接各系统模块之后，系统模型的创建工作便已结束。为了对动态系统进行正确的仿真与分析，必须设置正确的系统模块参数与系统仿真参数。系统模块参数的设置方法如下：

(1) 双击系统模块，打开系统模块的参数设置对话框。参数设置对话框包括系统模块的简单描述、模块的参数选项等信息。注意，不同系统模块的参数设置不同。

(2) 在参数设置对话框中设置合适的模块参数。根据系统的要求在相应的参数选项中设置合适的参数。

系统模块的参数设置如图 4.19 所示。

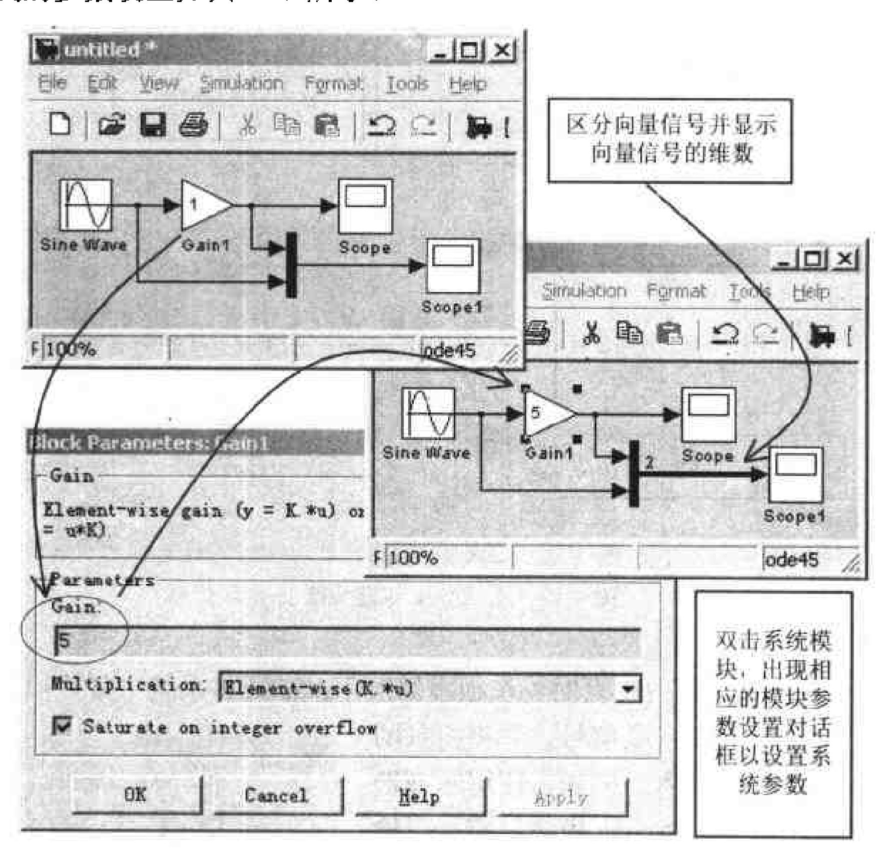


图 4.19 系统模块参数设置

当系统中各模块的参数设置完毕后，可设置合适的系统仿真参数以进行动态系统的仿真。有关系统仿真参数设置的知识将在第 5 章中进行详细的介绍，这里不再赘述。对于图 4.19 所示的动态系统，系统模块参数设置如图中所示（增益取值为 5），系统仿真参数采用 Simulink 的默认设置。

2. 运行仿真

当对系统中各模块参数以及系统仿真参数进行正确设置之后，单击系统模型编辑器上的 Play 图标（黑色三角）或选择 Simulation 菜单下的 Start 便可以对系统进行仿真分析。对于图 4.19 所示的动态系统，采用上述的模块参数设置与默认的仿真参数进行仿真。仿真结束后双击 Scope 模块以显示系统仿真的输出结果，如图 4.20 所示。

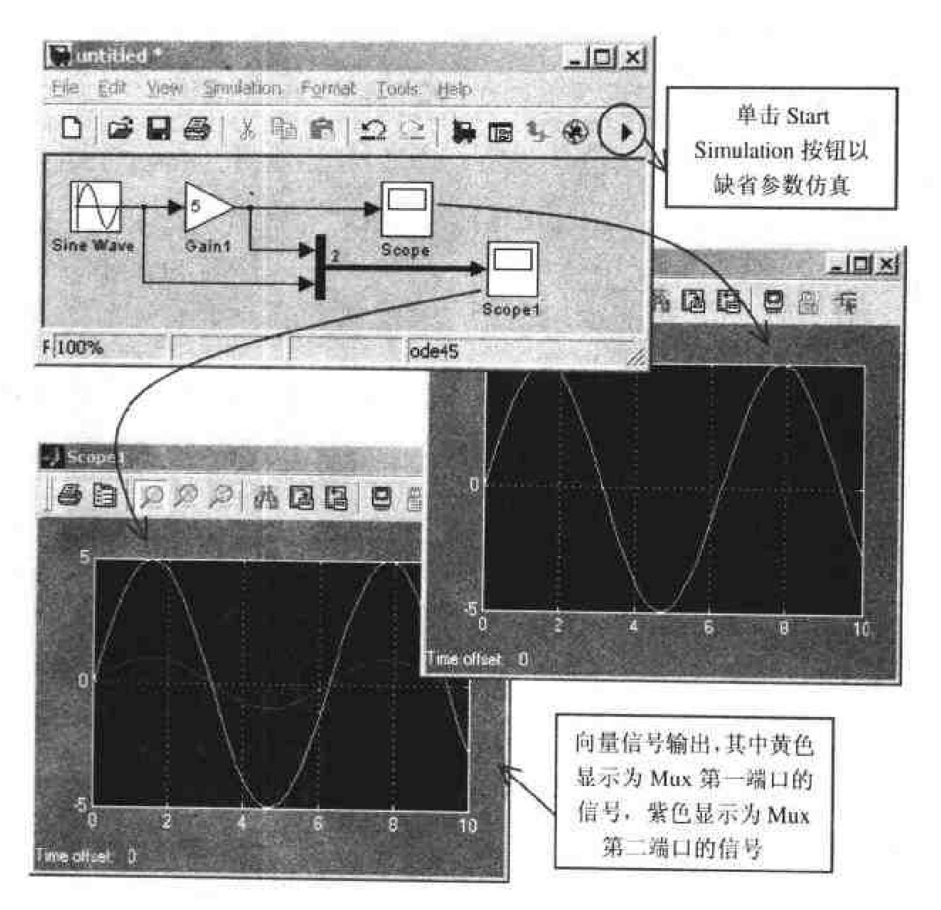


图 4.20　系统仿真及结果输出

本节以一个非常简单的动态系统为例说明如何利用Simulink构建用户自己的系统模型。系统模型框图的界面非常简单，无需进行特别设计；然而对于实际的、复杂的、大中型的动态系统而言，系统模型构建时的界面设计就显得非常重要。一个好的界面可以使系统模型的信号流程、输入输出关系一目了然，以便于用户的理解与维护。

4.4　设计 Simulink 框图的界面

4.3 节中对使用 Simulink 进行系统建模与仿真做了简单的介绍，任何动态系统的模型构建与仿真的步骤都与此类似。本节所要介绍的 Simulink 界面设计主要用来改善系统模型的界面，以便于用户对系统模型的理解与维护。

4.4.1　模块及框图属性编辑

1. 框图的视图调整

在 Simulink 系统模型编辑器中，可以对系统模型的视图进行调整以便更好地观察系统模型。视图调整的方法如下所述：

(1) 使用 View 菜单控制模型在视图区的显示，用户可以对模型视图进行任意缩放。

(2) 使用系统热键 R（放大）或 V（缩小）。

(3) 按空格键可以使系统模型充满整个视图窗口。

视图调整效果如图 4.21 所示。

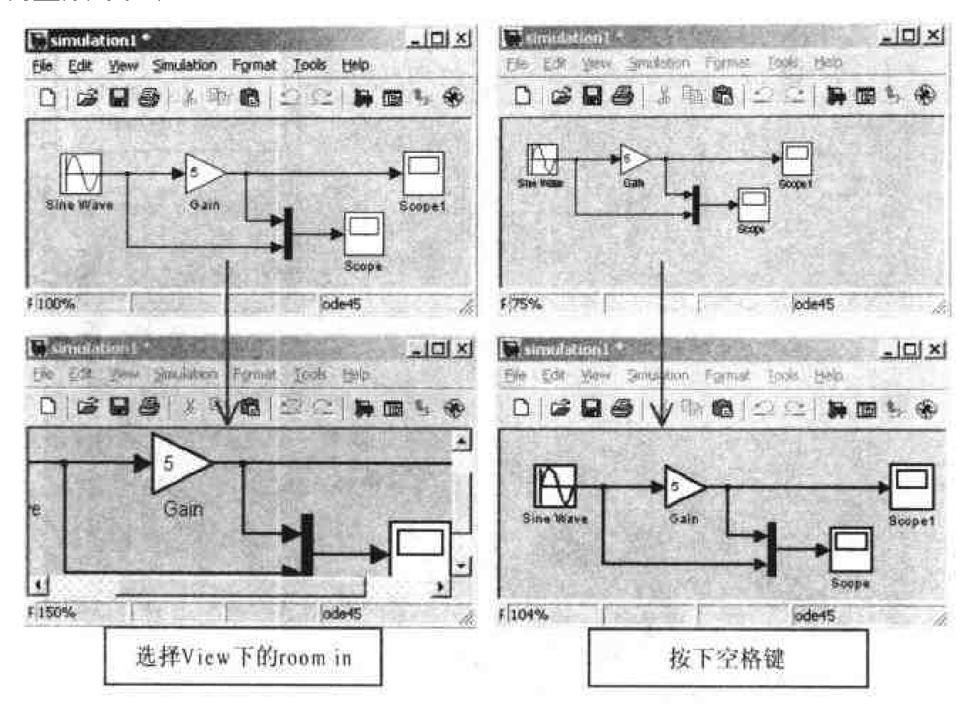


图 4.21　改变系统模型的视图

2. 模块的名称操作

在使用 Simulink 中的系统模块构建系统模型时，Simulink 会自动给系统模型中的模块命名，如在 4.3 节的简单动态系统中，正弦信号模块名称为 Sine Wave；对于系统模型中相同的模块，Simulink 会自动对其进行编号。一般对于简单的系统，可以采用 Simulink 的自动命名；但对于复杂系统，给每个模块取一个具有明显意义的名称非常有利于系统模型的理解与维护。下面简单介绍一下模块名称的操作。

(1) 模块命名：使用鼠标左键单击模块名称，进入编辑状态，然后键入新的名称。

(2) 名称移动：使用鼠标左键单击模块名称并拖动到模块的另一侧，或选择 Format 菜单中的 Flip Name 翻转模块名称。

(3) 名称隐藏：选择 Format 菜单中的 Hide Name 隐藏系统模块名称。

注意，系统模型中模块的名称应当是唯一的，否则 Simulink 会给出警告并自动改变名称。系统模型中模块的名称操作如图 4.22 所示。

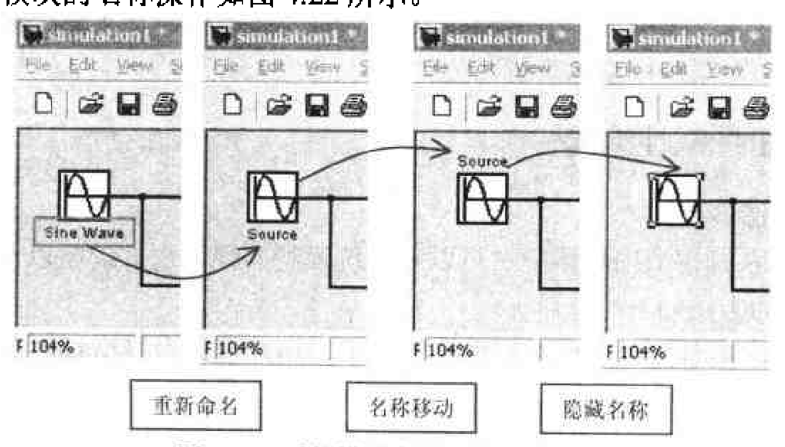


图 4.22　系统模型中模块的名称操作

3. 模块的其它操作

Simulink 允许用户对模块的几何尺寸进行修改，以改善系统模型框图的界面。例如，对于具有多个输入端口的模块，需要调整其大小使其能够较好地容纳多个信号连线，而非采用模块的默认大小；另外，对于某些系统模块，当模块的尺寸足够大时，模块的参数将直接显示在模块上面，这非常有利于用户对模型的理解。改变系统模块尺寸的方法为：使用鼠标左键单击选择模块，然后拖动模块周围任何一角的黑色方框到适当的大小。

Simulink 允许改变模块的颜色。使用鼠标右键单击模块，选择 Foreground color 或 Background color 菜单来设置颜色；或使用模型编辑器中 Format 菜单中的相应命令设置模块颜色。如果模块的前景色发生改变，则所有由此模块引出的信号线颜色也随之改变；当系统模型框图很复杂时，这个特性能够有效地增强框图的可读性。

此外，还可以使用 Format 菜单中的 Show Drop Shadow 为模块生成阴影，或使用 Flip Block、Rotate Block 对模块进行翻转与旋转，或使用 Font 对模块字体进行设置等。对模块的操作如图 4.23 所示。

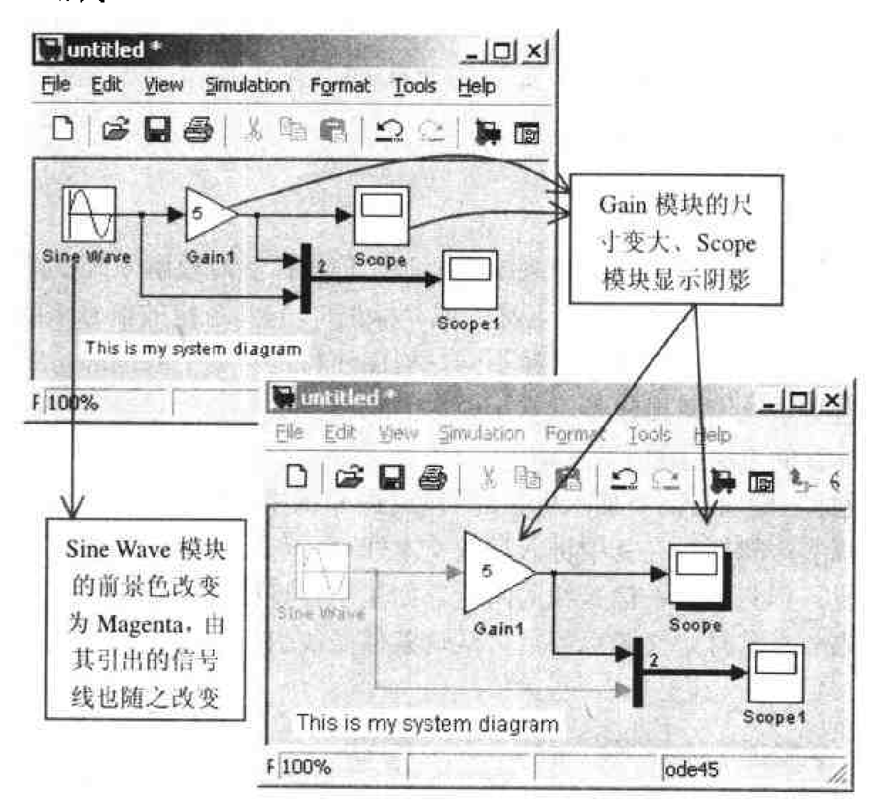


图 4.23　模块的其它操作

4. 系统框图注释

作为友好的 Simulink 系统模型界面，对系统模型的注释是不可缺少的。在 Simulink 中对系统模型框图进行注释的方法非常简单，只需在系统模型编辑器的背景上双击鼠标左键以确定添加注释文本的位置，并打开一个文本编辑框，用户便可以在此输入相应的注释文本。输入完毕后，使用鼠标左键单击以退出编辑并移动文本位置（编辑框未被选中情况下）到合适的地方。此外，在文本对象上单击鼠标右键，可以改变文本的属性如大小、字体和对齐方式等。在任何时候都可以双击注释文本进行编辑。系统框图注释如图 4.24 所示。

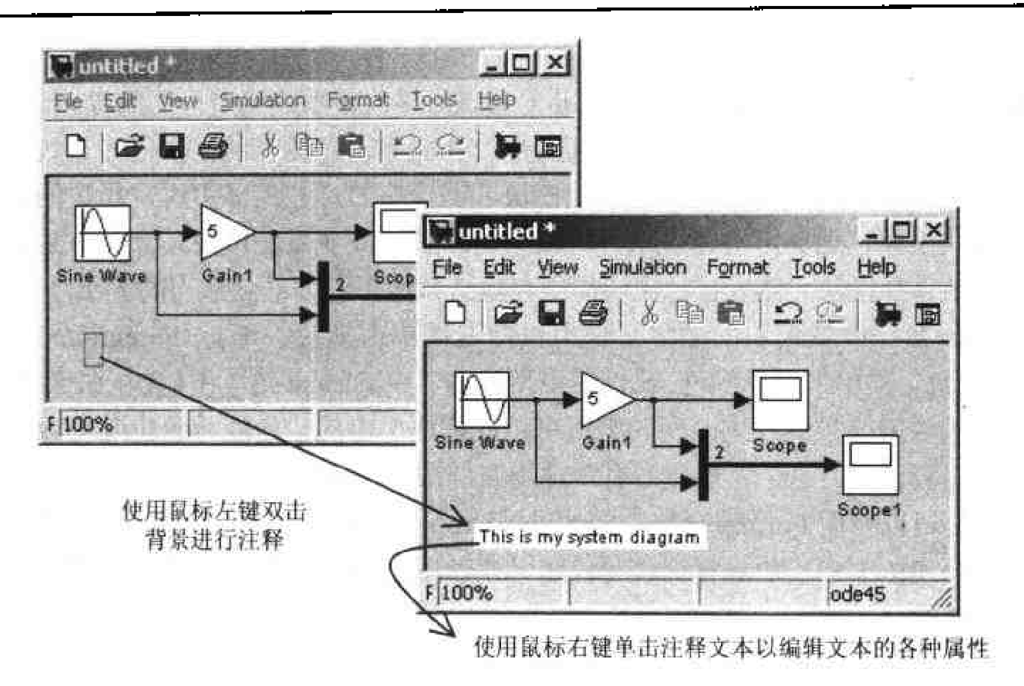


图 4.24　系统模型框图注释

4.4.2　信号标签与标签传递

1. 信号标签

在创建系统模型尤其是大型复杂系统模型时，信号标签对理解系统框图尤为重要。所谓的信号标签，也可以称为信号的“名称”或“标记”，它与特定的信号相联系，是信号的一个固有属性。这一点与系统框图注释不同，框图注释是对整个或局部系统模型进行说明的文字信息，它与系统模型相分离。

生成信号标签的方法有如下两种：

(1) 使用鼠标左键双击需要加入标签的信号（即系统模型中与信号相对应的模块连线），这时便会出现标签编辑框，在其中键入标签文本即可。与框图注释类似，信号标签可以移动到希望的位置，但只能是在信号线的附近。如果强行将标签拖动离开信号线，标签会自动回到原处。当一个信号定义了标签后，从这条信号线引出的分支线会继承这个标签，如图 4.25 所示。

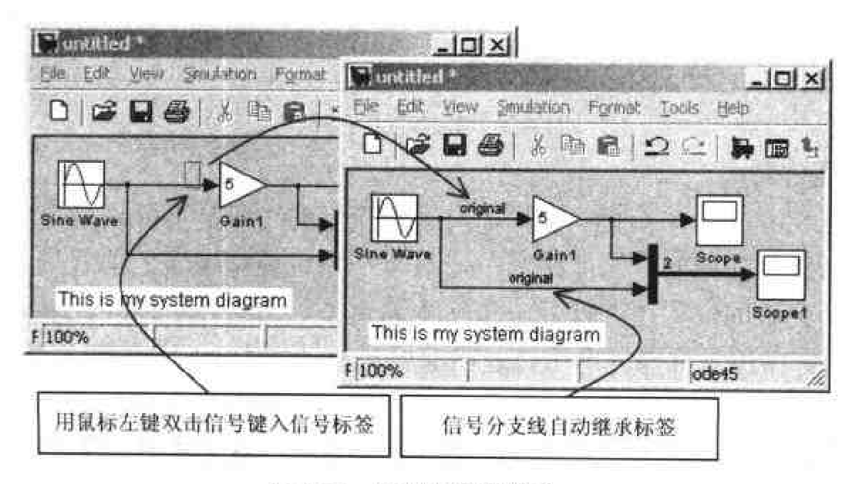


图 4.25　信号标签操作之一

(2) 首先选择需要加入标签的信号，用鼠标左键单击信号连线；然后使用 Edit 菜单下的 Signal Properties 项，在打开的界面中编辑信号的名称，而且还可以使用这个界面对信号作简单的描述并建立 HTML 文档链接，如图 4.26 所示。

注意，虽然信号标签的内容可以任意指定，但为了系统模型可读性，信号标签最好使用能够代表信号特征的名称（如信号类型、信号作用等）。

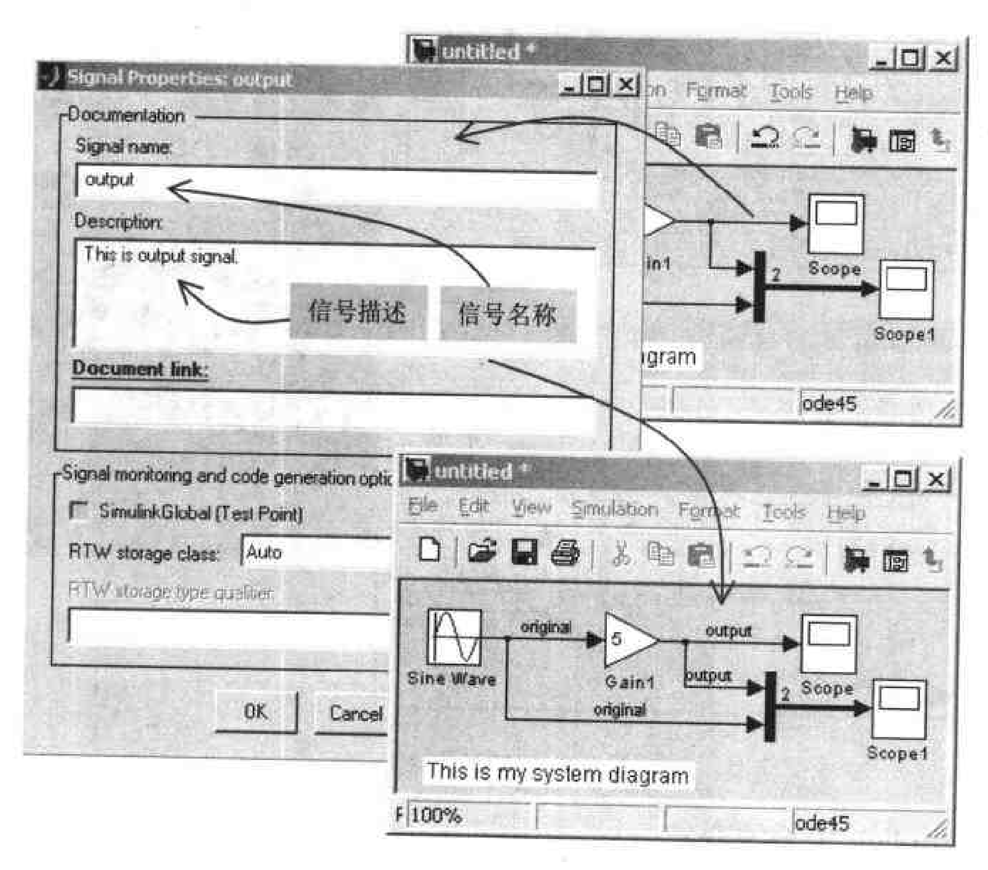


图 4.26　信号标签操作之二

2. 信号标签的传递

在系统模型中，信号标签可以由某些称之为"虚块"的系统模块来进行传递。这些虚块主要用来完成对信号的选择、组合与传递，它不改变信号的任何属性。如 Signals & Systems 模块库中的 Mux 模块的功能是组合信号，但并不改变信号的值。

信号标签传递的方法有如下几种：

(1) 选择信号线并用鼠标左键双击，在信号标签编辑框中键入< >,在此尖括号中键入信号标签即可传递信号标签。然后选择 Edit 菜单中的 Update Diagram 刷新模型。

(2) 选择信号线，然后选择 Edit 菜单中的 Signal Properties；或单击鼠标右键，选择弹出式菜单中的 Signal Properties，将 Show Propagated Signals 设置为 on 即可。注意：只能在信号的前进方向上传递该信号标签。当一个带有标签的信号与 Scope 块连接时，信号标签将作为标题显示。信号标签的传递如图 4.27 所示。

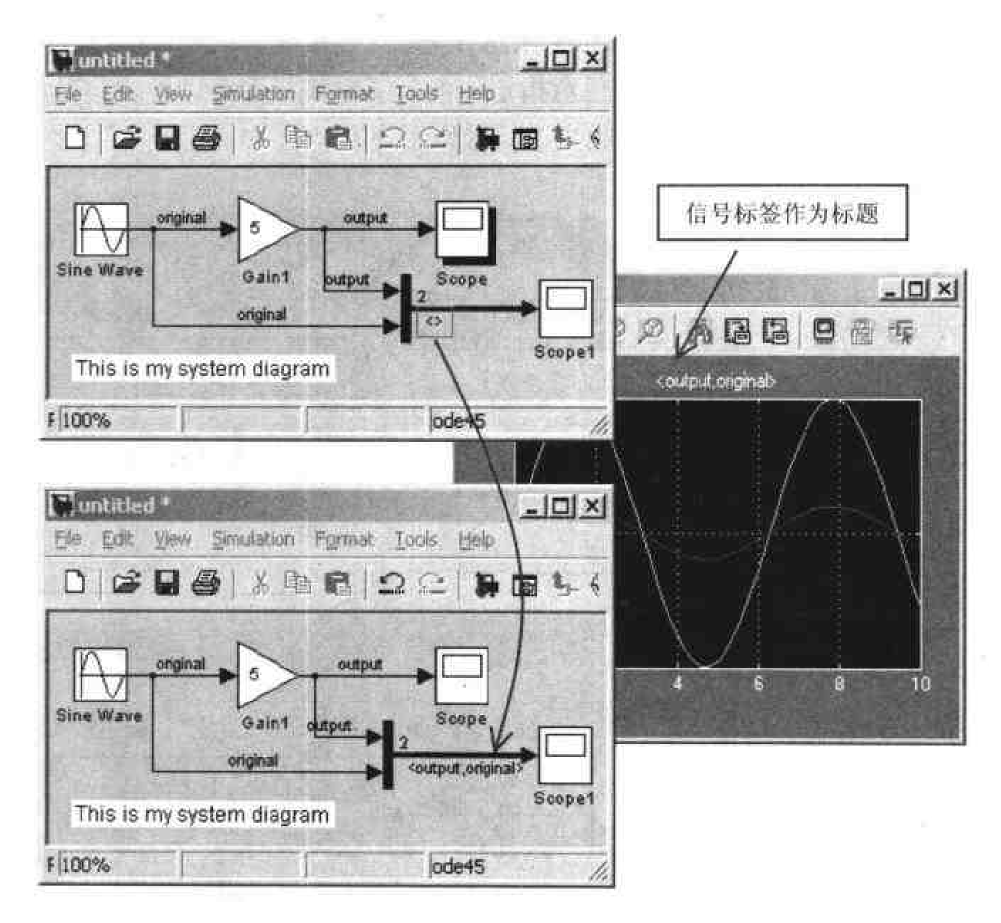


图 4.27 信号标签的传递

4.4.3 Simulink 子系统介绍

对于简单的动态系统而言，用户很容易建立系统模型并分析系统模型中各模块之间的相互关系，以及模块的输入输出关系。但是对于比较复杂的系统，系统模型中包含的模块数目较多，模块之间的输入输出关系比较复杂。这时对于分析与设计系统而言，都会给用户带来诸多的不便，而使用子系统技术则可以较好地解决这一问题。

1. 子系统生成

Simulink 提供的子系统功能可以大大地增强 Simulink 系统模型框图的可读性。所谓的子系统可以理解为一种“容器”，此容器能够将一组相关的模块封装到一个单独的模块中，并且与原来系统模块组的功能一致。

子系统的建立方法有如下两种：

(1) 在已有的系统模型中建立子系统：首先框选待封装的区域，即在模型编辑器背景中单击鼠标左键并拖动，选中需要放置到子系统中的模块与信号（或在按下 Shift 键的同时，用鼠标左键单击所需模块）；然后选择 Edit 菜单下的 Create Subsystem，即可建立子系统。如图 4.28 所示。

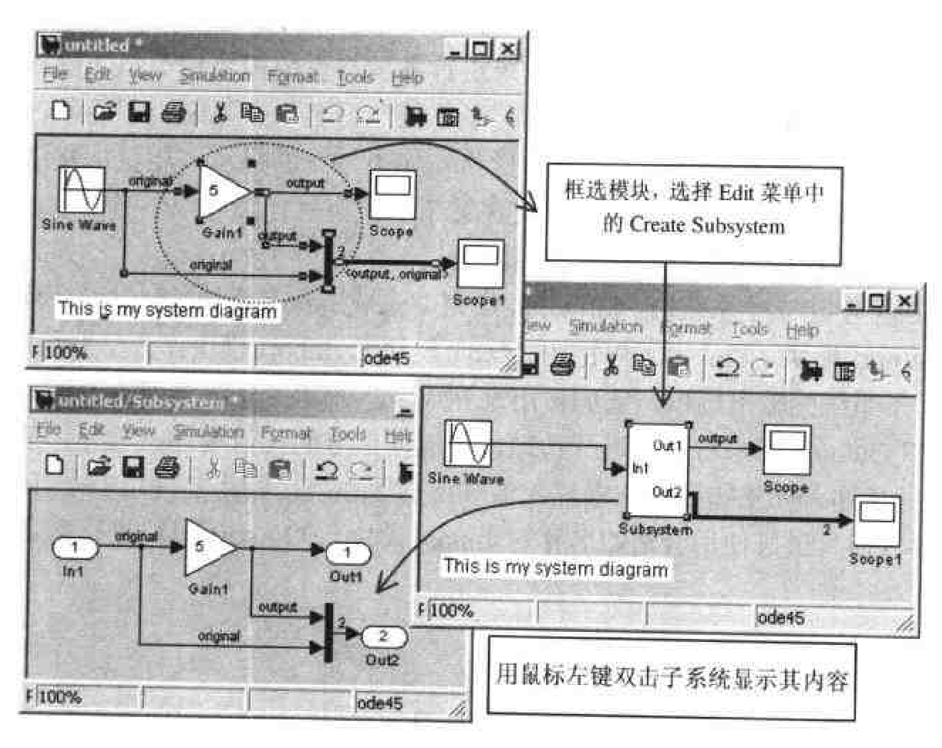


图 4.28　子系统建立：选择模块生成子系统

(2) 建立空的子系统：使用 Subsystems 模块库中的模块建立子系统。这样建立的子系统内容为空，然后双击子系统对其进行编辑。如图 4.29 所示。

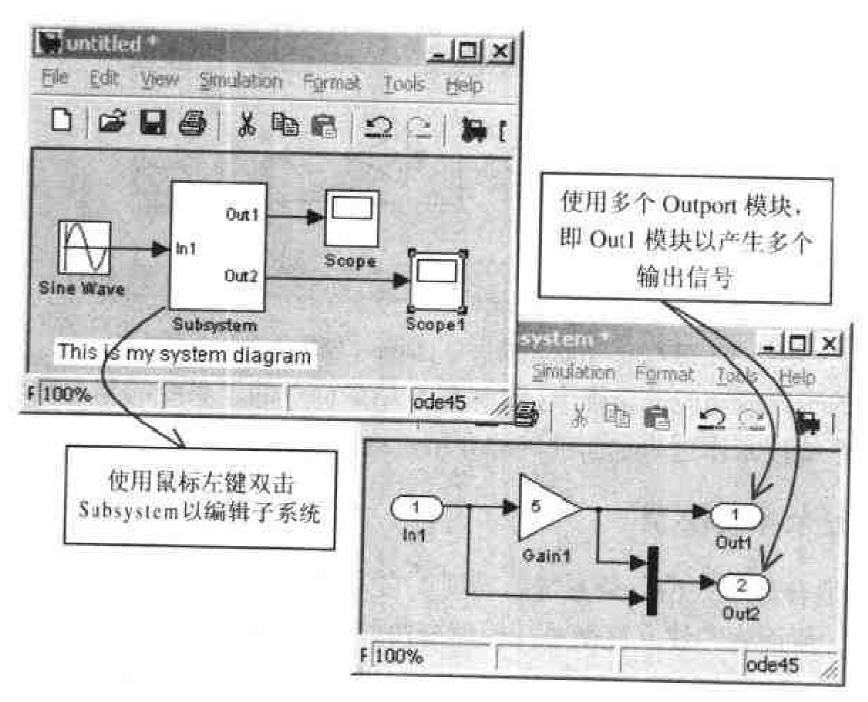


图 4.29　子系统建立：生成并编辑空子系统

建立此系统模型所需要的系统模块如下所述：Subsystems 模块库中的 Subsystem 模块、Sources 模块库中的 Sine Wave、Sinks 模块库中的 Scope 模块、Sinks 模块库中的 Out1 模块（Subsystem 模块的缺省设置为单输入单输出，使用 Out1 模块可以产生多个输出）、Math 模块库中的 Gain 模块以及 Signals & Systems 模块库中的 Mux 模块等。

2. 子系统操作

在生成子系统之后，用户可以对子系统进行各种与系统模块相类似的操作，这时子系统相当于具有一定功能的系统模块。例如，子系统的命名、子系统视图的修改、子系统的显示颜色等等。当然子系统也有其特有的操作，如子系统的显示（用鼠标左键双击子系统模块即可打开子系统）、子系统的封装（将在第 7 章中进行详细介绍）等等。

3. Inport 输入模块与 Outport 输出模块

在系统模型中建立子系统时，Simulink 会自动生成 Inport 模块（Sources 模块库中的 In1 模块）与 Outport 模块（Sinks 模块库中的 Out1 模块）。Inport 模块作为子系统的输入端口，Outport 作为子系统的输出端口，它们被用来完成子系统和主系统之间的通讯。

Inport 和 Outport 用来对信号进行传递，不改变信号的任何属性；另外，信号标签可以越过它们进行传递。如果需要建立多输入多输出的子系统，则需要使用多个 Inport 模块与 Outport 模块，而且最好使用合适的名称对 Inport 模块与 Outport 模块进行命名，如图 4.30 所示。

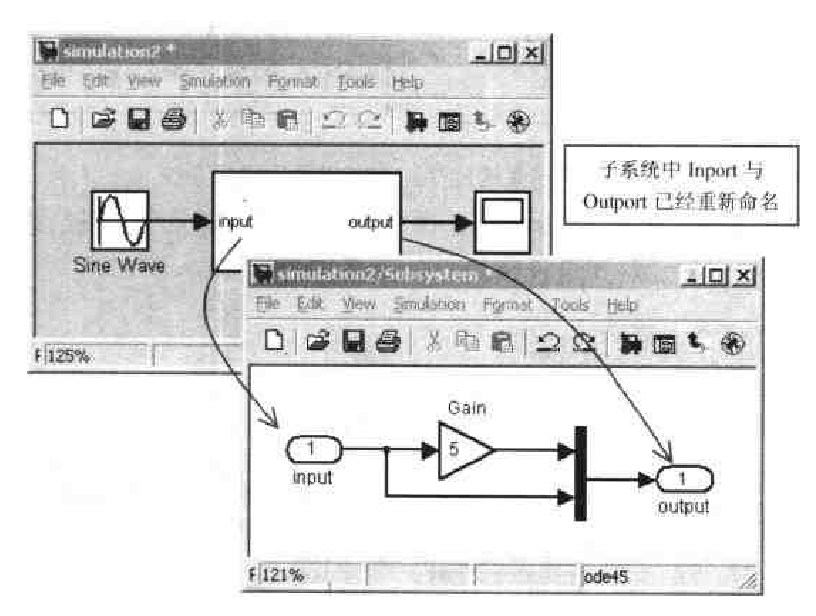


图 4.30 Inport 模块与 Outport 模块的重新命名

Simulink 的子系统功能非常强大，这里仅介绍最简单的子系统的基本概念，第 7 章将对 Simulink 的子系统技术作更为全面细致的介绍，这里不再赘述。

4.4.4 建立复杂系统模型

Simulink 适合建立大型复杂系统的模型，它为仿真系统模型的界面组织与设计提供了强大的支持。一般而言，建立复杂系统模型有两种不同的思路：

(1) 自下向上的设计思路：如果用户从草图开始建立一个复杂的模型，可以先建底层模型，然后对已经建好的块生成子系统。

(2) 自顶向下的设计思路：首先设计系统的总体模型，然后再进行细节设计。采用这种方法，可以在顶层使用空的子系统块，然后再实现具体的细节。这种方法需要对系统各部分的功能有比较清楚的理解。

对于简单的系统模型，用户可以直接对其进行管理与维护。但是对于复杂系统模型而言，使用 Simulink 模型浏览器能够帮助用户更好的组织、管理与理解系统模型。模型浏览器允许用户分层次地显示系统模型，只需要单击层中的某个模块便可观察其中的内容。选择 View 菜单下的 Model Browser Options\Show Model Browser 可以激活模型浏览器。这里给出一个复杂系统的框图作为说明（MATLAB 控制工具箱中的例子 rolling_mill.mdl），如图 4.31 所示。

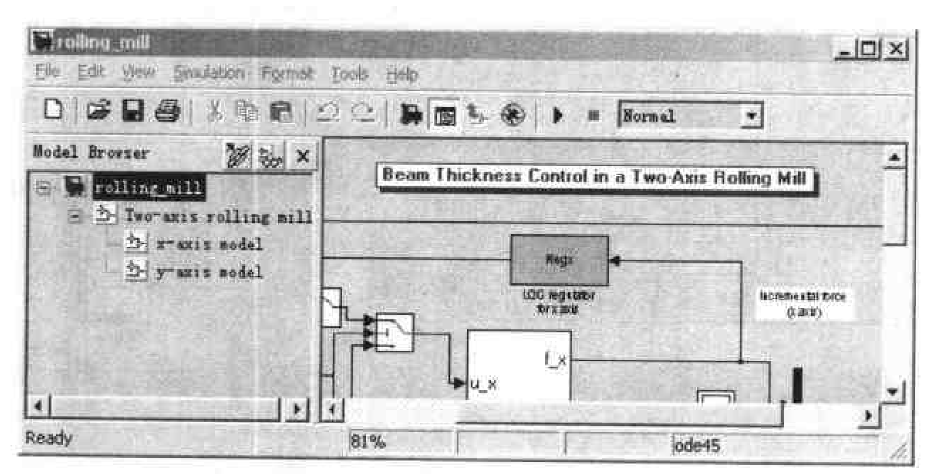


图 4.31　模型浏览器的使用

对于复杂系统模型的建立，用户一定要对系统的总体结构作一个了解，然后利用不同的设计思路进行设计，并在生成或封装子系统之前，对其作详细的测试。这对以后的设计有很大的帮助。

4.5　Simulink 与 MATLAB 的接口设计

Simulink 是基于 MATLAB 平台之上的系统级仿真平台，它与 MATLAB 紧密地集成在一起。Simulink 不仅能够采用 MATLAB 的求解器对动态系统进行求解，而且还可以和 MATLAB 进行数据交互（从 MATLAB 的工作空间中读入数据或者向 MATLAB 工作空间中写入数据）。本节将对 Simulink 与 MATLAB 的接口（也就是数据交互）作简单的介绍。

4.5.1　由 MATLAB 工作空间变量设置系统模块参数

如前所述，用户可以双击一个模块以打开模块参数设置对话框，然后直接输入数据以设置模块参数。其实，用户也可以使用 MATLAB 工作空间中的变量设置系统模块参数，这对于多个模块的参数均依赖于同一个变量时非常有用。由 MATLAB 工作空间中的变量设置模块参数的形式有如下两种：

(1) 直接使用 MATLAB 工作空间中的变量设置模块参数。

(2) 使用变量的表达式设置模块参数。

例如，如果 a 是定义在 MATLAB 中的变量，则表达式 a、a^2+5、exp(-a)等均可以作为系统模块的参数，如图 4.32 所示。

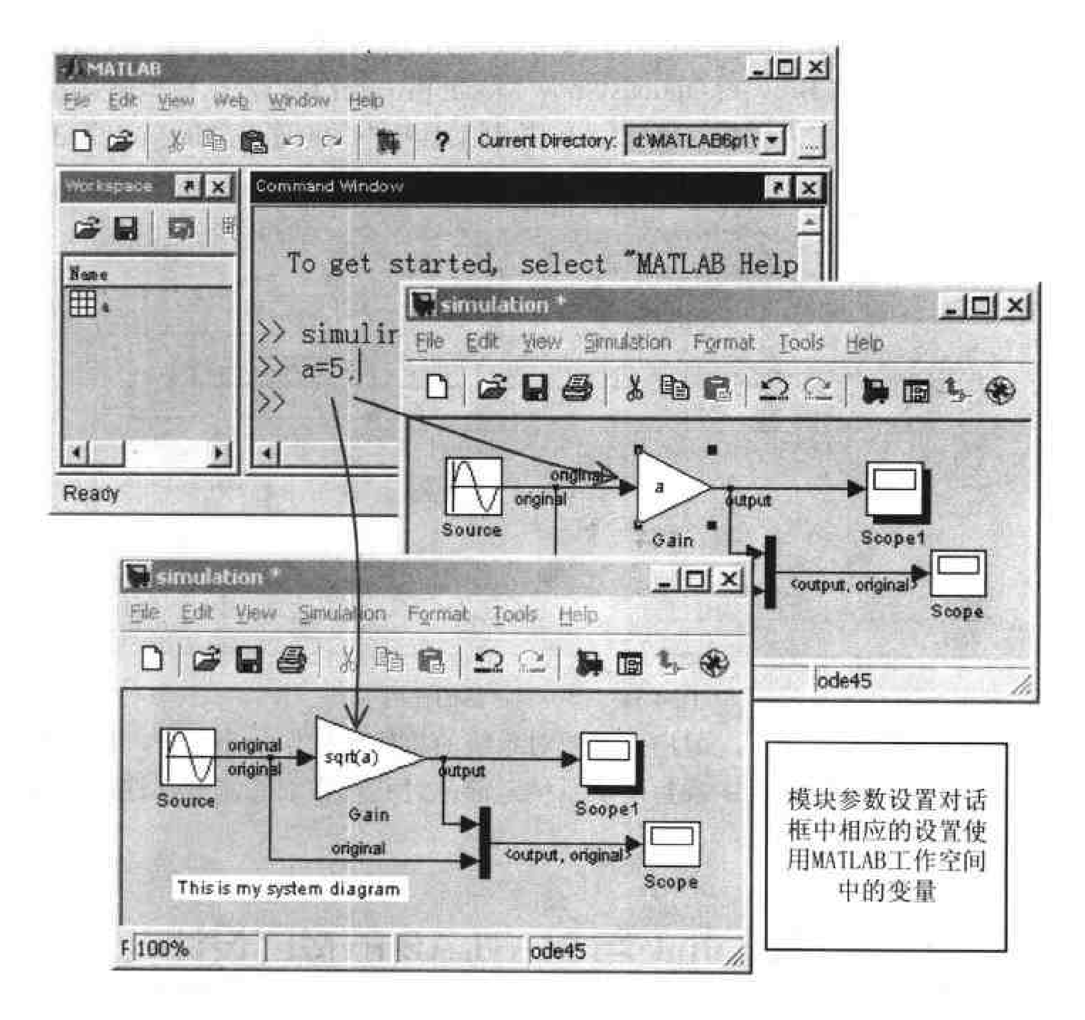


图 4.32　使用 MATLAB 工作空间变量设置模块参数

注意：如果系统模块参数设置中使用的变量在 MATLAB 工作空间中没有定义，仿真开始的时候会出现错误信息。

4.5.2　将信号输出到 MATLAB 工作空间中

使用示波器模块 Scope 的输出信号，可以使用户对输出的信号进行简单的定性分析。然而，有时需要对输出信号作定量的分析，此时用户可以先将系统模型中的选定信号输出到 MATLAB 工作空间中，然后再作进一步的定量分析。

使用 Sinks 模块库中的 To Workspace 模块，可以轻易地将信号输出到 MATLAB 工作空间中。信号输出的名称在 To Workspace 模块的对话框中设置，此对话框还可以设置输出数据的点数、输出的间隔，以及输出数据的类型等。其中输出类型有三种形式：数组、结构以及带有时间变量的结构。仿真结束或暂停时信号被输出到工作空间中，如图 4.33 所示。

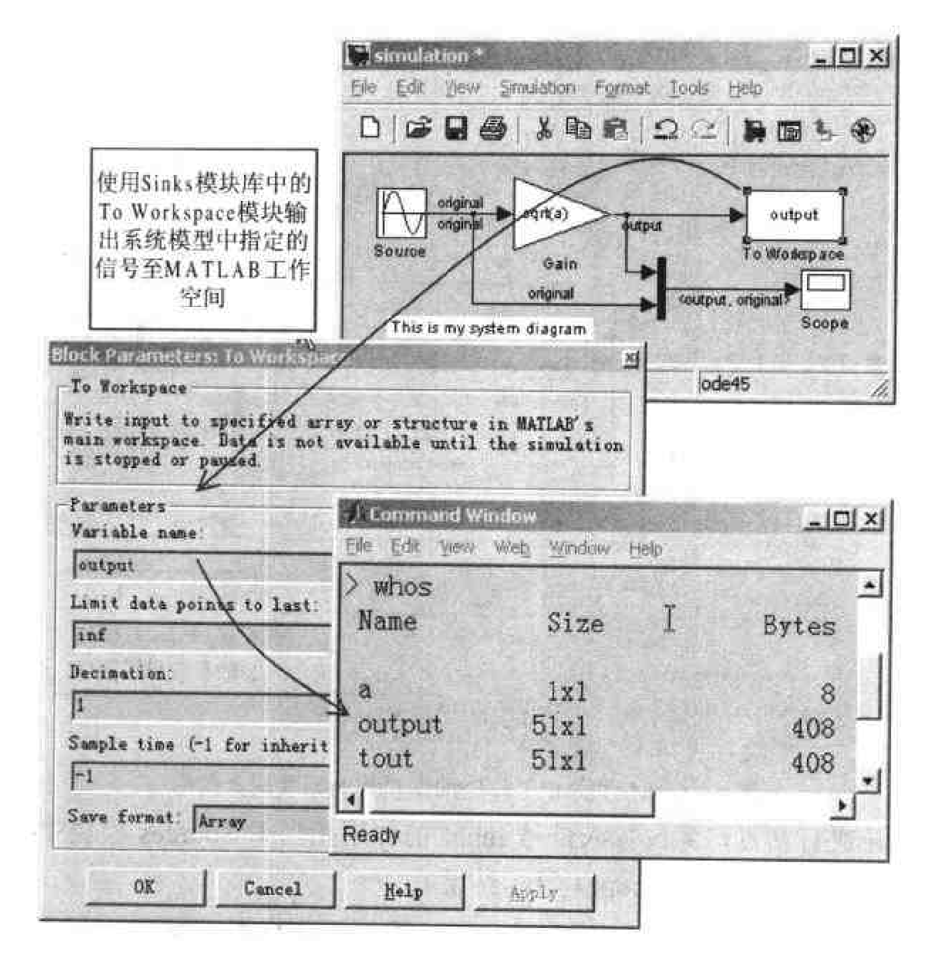


图 4.33　系统模型中信号输出

4.5.3　使用工作空间变量作为系统输入信号

Simulink 与 MATLAB 的数据交互是相互的，除了可以将信号输出到 MATLAB 工作空间中之外，用户还可以使用 MATLAB 工作空间中的变量作为系统模型的输入信号。使用 Sources 模块库中的 From Workspace 模块可以将 MATLAB 工作空间中的变量作为系统模型的输入信号。此变量的格式如下所示：

```
>>t=0:time_step:final_time;        % 表示信号输入时间范围与时间步长
>>x=func(t);                       % 表示在每一时刻的信号值
>>input=[t',x'];
%表示信号的输入向量，输入变量第一列须为时间序列，接下来的各列代表信号的取值
```

例如，在 MATLAB 命令窗口中键入如下的语句并运行。

```
>>t=0:0.1:10;
>>x=sin(t);
>>input=[t',x'];
```

在系统模型的 From Workspace 中使用此变量作为信号输入，如图 4.34 所示。

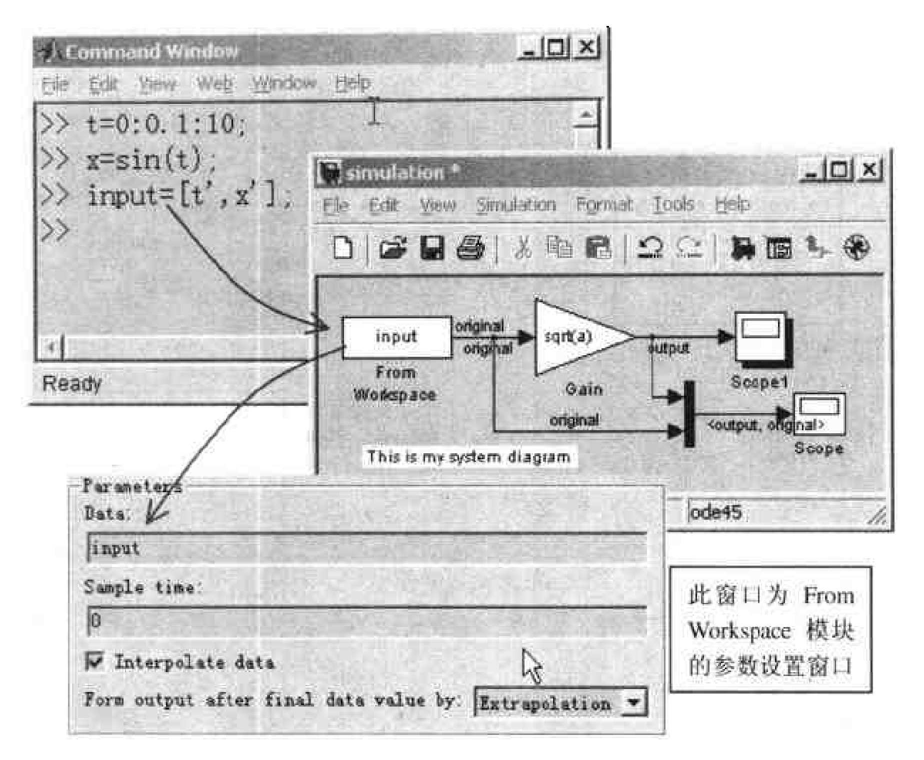


图 4.34 MATLAB 工作空间变量作为系统输入信号

运行此系统进行仿真，系统输入信号 input 的作用相当于 Sources 模块中的 Sine Wave 模块，其结果如图 4.35 所示（Scope1 显示结果）。

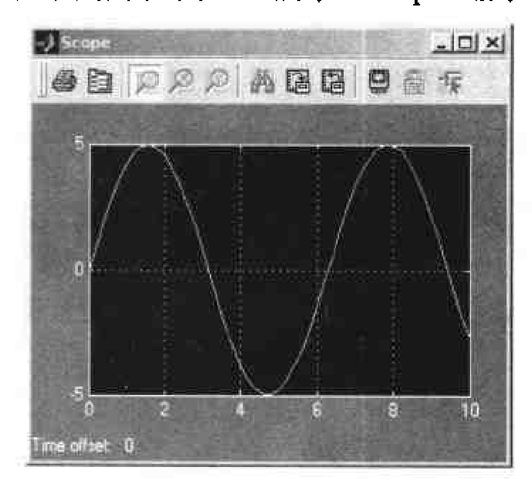


图 4.35 使用 input 信号作为输入的仿真结果

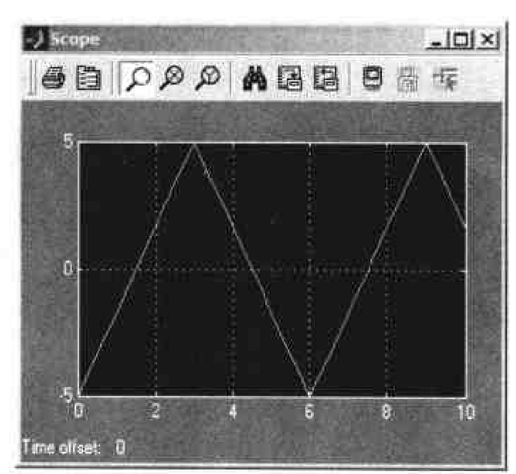


图 4.36 三角波输入的仿真结果

注意：在必要的情况下，Simulink 会对没有定义的时间点进行线性插值。例如：

```
>> t=[0 3 6 9 10];
>> x=[-1 1 -1 1 1/3];
>> input=[t',x'];
```

将生成一个三角波。采用此信号对上述系统进行仿真时，仿真输出结果如图 4.36 所示。显然，Simulink 对 MATLAB 工作空间的输入信号 input 进行了线性插值。

4.5.4　向量与矩阵

在前面的系统模型中，Simulink 所使用的信号均是标量。其实，Simulink 也能够传递和使用向量信号。例如，向量增益可以作用在一个标量信号上，产生一个向量输出。在缺省情况下，模块对向量中的逐个元素进行操作，就像 MATLAB 中的数组运算一样，如图 4.37 所示。

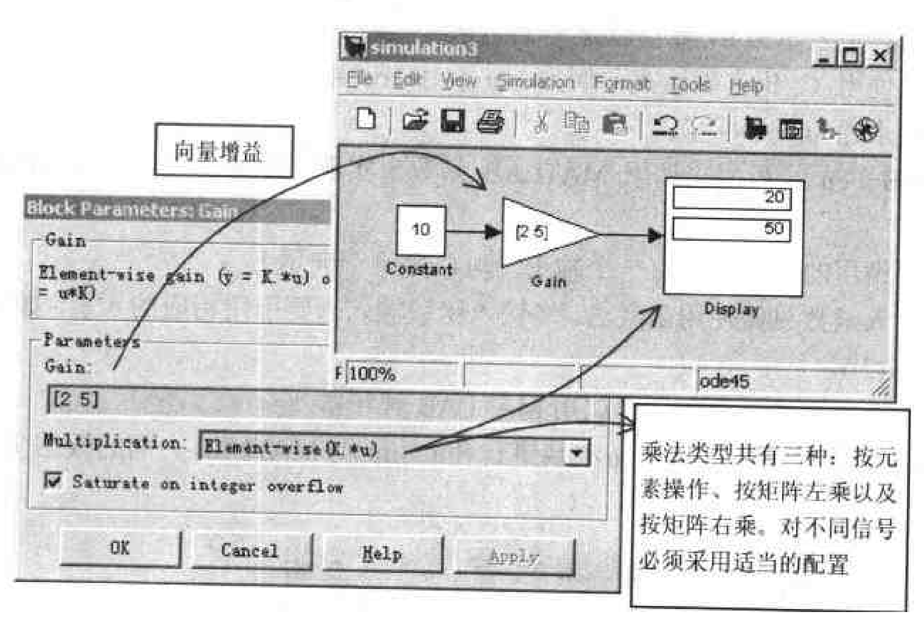


图 4.37　向量增益示意图

Simulink 4 最重要的特性就是支持矩阵形式的信号，它可以区分行和列向量并传递矩阵。通过对模块做适当的配置，可以使模块能够接受矩阵作为模块参数。在上面的例子中，如果 Constant 模块的参数为一矩阵，并且 Gain 增益模块被配置成按矩阵乘的定义从左边乘上输入向量，则 Display 块能够感知到输入信号的尺寸，即 1×2 行向量，并对边框做适当调整，如图 4.38 所示。

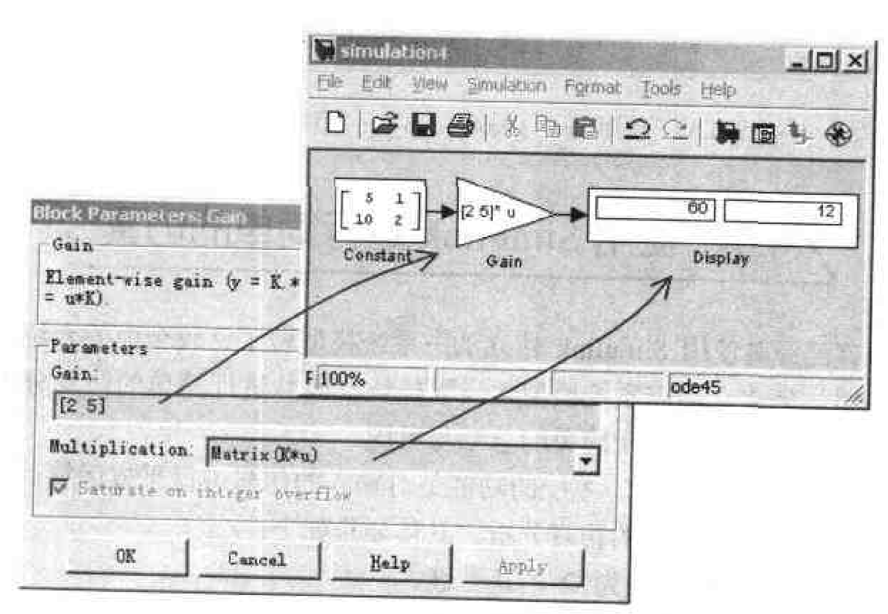


图 4.38　矩阵输入与向量增益示意图

4.5.5 MATLAB Function 与 Function 模块

除了使用上述的方式进行 Simulink 与 MATLAB 之间的数据交互，用户还可以使用 Functions and Tables 模块库中的 Function 模块（简称为 Fcn 模块）或 Functions and Tables 模块库中的 MATLAB Function 模块（简称为 MATLAB Fcn 模块）进行彼此间的数据交互。

Fcn 模块一般用来实现简单的函数关系，在 Fcn 模块中：

(1) 输入总是表示成 u，u 可以是一个向量。

(2) 可以使用 C 语言表达式，例如 sin(u[1])+cos(u[2])。

(3) 输出永远为一个标量。

MATLAB Fcn 一般用来调用 MATLAB 函数来实现一定的功能，在 MATLAB Fcn 模块中：

(1) 所要调用的函数只能有一个输出（可以是一个向量）。

(2) 单输入函数只需使用函数名，多输入函数输入需要引用相应的元素，如 mean、sqrt、myfunc(u(1),u(2))。

(3) 在每个仿真步长内都需要调用 MATLAB 解释器。

使用 Fcn 模块与 MATLAB Fcn 模块进行 Simulink 与 MATLAB 之间的数据交互如图 4.39 所示。

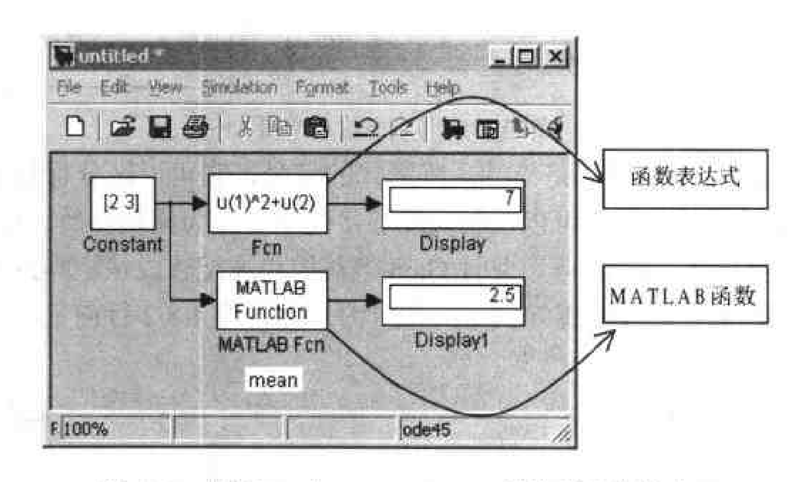


图 4.39 使用 Fcn 与 MATLAB Fcn 模块进行数据交互

4.6 使用 Simulink 进行简单的仿真

前面的内容已经对使用 Simulink 建立动态系统模型做了比较全面的介绍，本节将使用具体的例子说明如何建立动态系统模型、设置模块参数并进行简单的仿真分析，从而对使用 Simulink 建立系统模型作一个简单的总结与巩固。

【例 4.1】 信号平方运算。系统的功能是对输入的信号进行平方运算。现要求建立系统的 Simulink 模型并进行简单的仿真分析。具体要求如下：

(1) 系统输入信号源：幅值为 2 的正弦波。

(2) 使用 Scope 显示原始信号和结果信号。

(3) 生成系统运算部分的子系统。

(4) 生成信号标签并传递。

解：首先选择系统所需的如下模块（组件）：

(1) Sources 库中的 Sine Wave 块。

(2) Math 库中的 Product 块。

(3) Signals and Systems 库中的 Mux 块。

(4) Sinks 库中的 Scope 块。

然后进行如下的操作：

(1) 连接系统模块。

(2) 选择一个包含 Product 和 Mux 块的区域，建立相应的子系统。

(3) 在主系统中生成输入信号的标签，在子系统中生成输出信号的标签。

(4) 传递信号的标签。

(5) 改变输入和输出端口的名字。

(6) 保存模型。

建立好的系统模型与相应的子系统实现如图 4.40 所示（在系统模型中，我们改变了所有系统模块的字体以获得更好的视觉效果）。

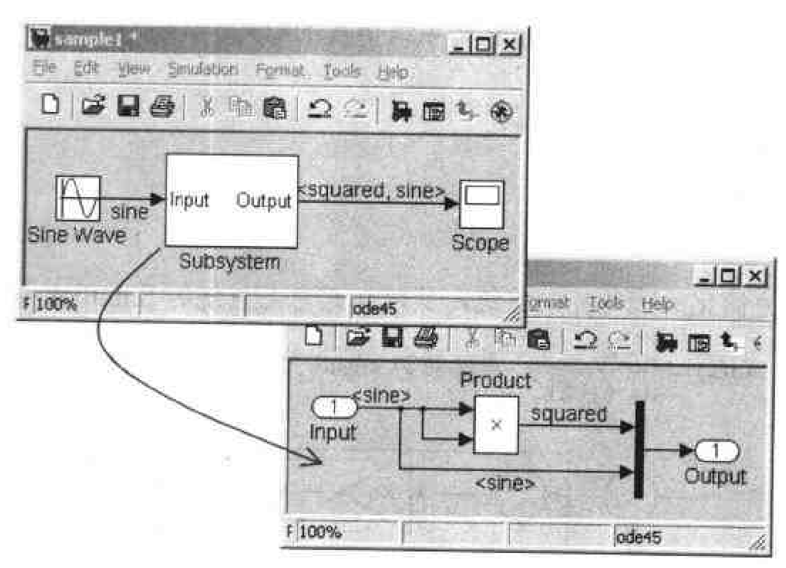


图 4.40 平方运算系统模型

最后，进行模块参数设置并使用默认的仿真参数进行仿真。在本例中，只需要对系统输入信号源 Sine Wave 模块进行参数设置即可（双击 Sine Wave 模块），设置正弦信号的幅值为 2，如图 4.41 所示。系统仿真结果如图 4.42 所示。

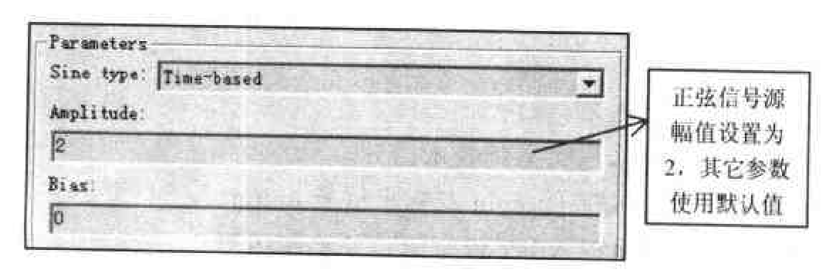


图 4.41 Sine Wave 模块参数设置

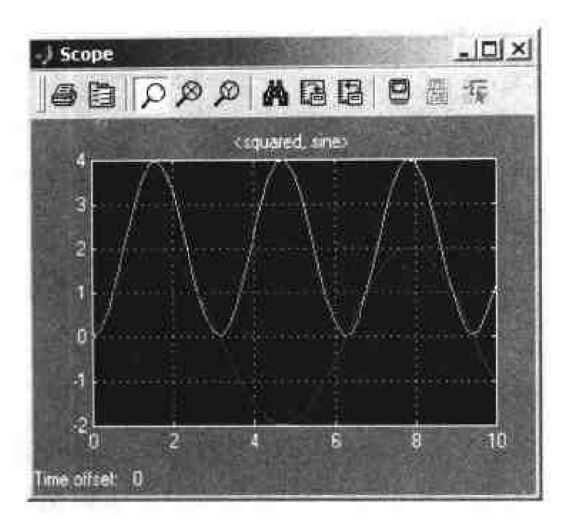


图 4.42 系统仿真结果

本章对 Simulink 的模型构建作了一个比较全面的介绍，对于不同领域的工程技术人员，都可以利用本章介绍的基本知识对大部分的系统进行建模与简单分析。至此，用户应该能够熟练快速地建立自己的系统模型。Simulink 的功能非常强大，它可以大大提高系统设计、仿真与分析的效率。本章是 Simulink 最基础的知识。后面的章节将详细介绍使用 Simulink 进行系统仿真的高级技术，以及 Simulink 系统仿真的原理。



习 题

图 4.43 所示为简化的飞行控制系统，试建立此动态系统的 Simulink 模型并进行简单的仿真分析。其中，$G(s)=\dfrac{25}{s(s+0.8)}$，系统输入 input 为单位阶跃响应，Ka = 2，Kb = 1。

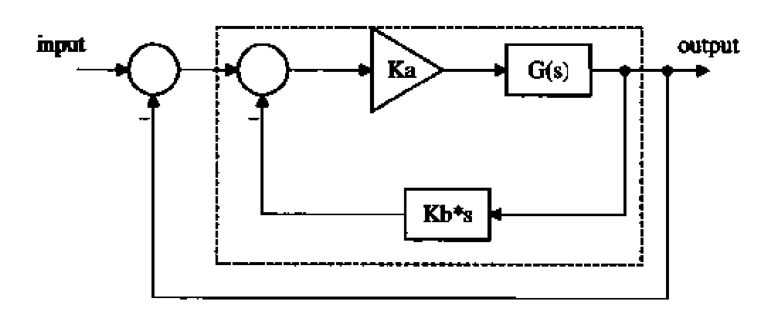


图 4.43 简化后的飞行控制系统

具体要求如下：

(1) 采用自顶向下的设计思路。

(2) 对虚线框中的控制器采用子系统技术。

(3) 用同一示波器显示输入信号 input 与输出信号 output。

(4) 输出数据 output 到 MATLAB 工作空间，并绘制图形。

注 1：解答提示。

选择如下合适的系统模块（组件）：

主系统所需模块：

(1) Sources 模块库的 Step 阶跃输入模块。
(2) Sinks 模块库中的 Scope 模块。
(3) Sinks 模块库中的 To Workspace 模块。
(4) Subsystems 模块库中的 Subsystem 模块。
(5) Signals & Systems 模块库中的 Mux 模块。
(6) Math 模块库中的 Sum 模块。

子系统所需模块：

(7) Continuous 模块库中 Derivative 模块。
(8) Continuous 模块库中的 Zero－Pole 模块。
(9) Math 模块库中的 Gain 模块、Sum 模块。

相应的操作：

(1) 连接主系统中各模块。
(2) 双击子系统，对空的子系统进行编辑。
(3) 配置子系统参数。
(4) 配置主系统参数。
(5) 保存模型，运行系统，进行仿真。
(6) 在 MATLAB 中绘制输出信号。

注 2：一些说明。

(1) 在控制器中 Kb*s 模块的实质为微分，故使用 Derivative 模块实现。

(2) G(s)采用零极点模型表示，故使用 Zero–Pole 模块，由于 G(s)无零点，在其设置中用[ ]表示。

注 3：参考模型框图及部分参数设置如图 4.44、4.45 所示。

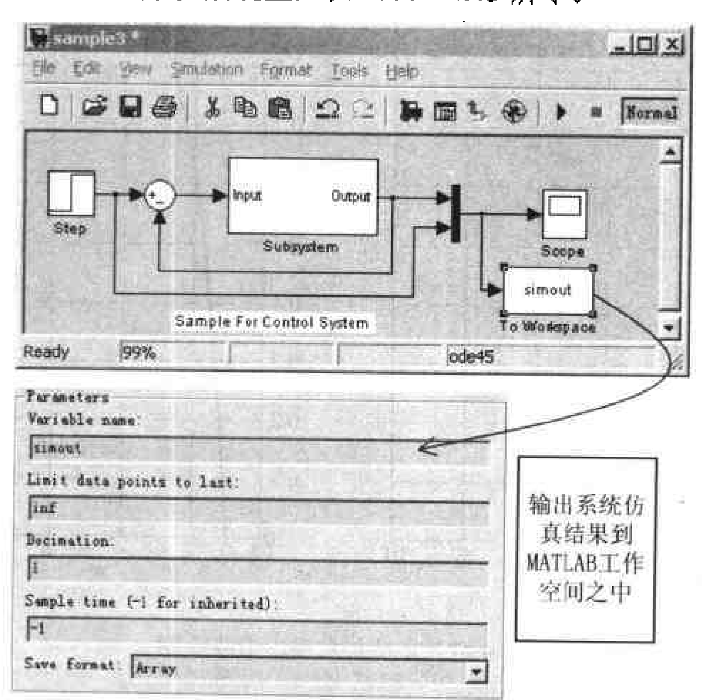


图 4.44　主系统模型及 To Workspace 模块参数设置

注 4：系统仿真结果，如图 4.46 所示。

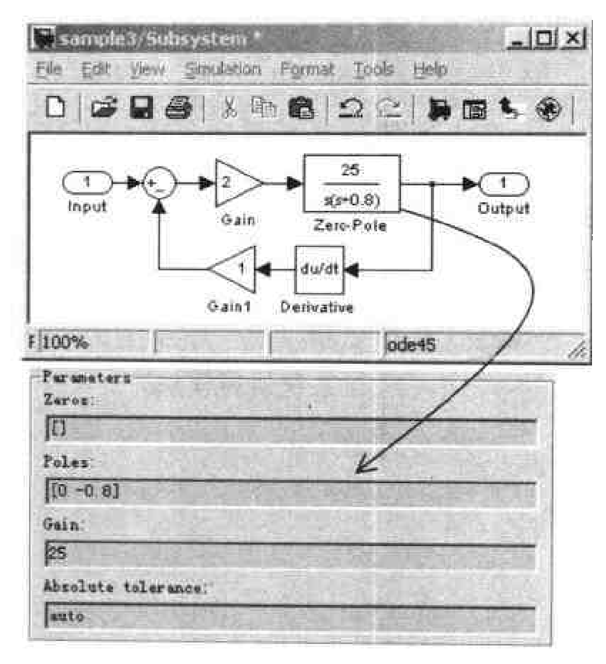


图 4.45　子系统模型及 Zero–Pole 模块参数设置

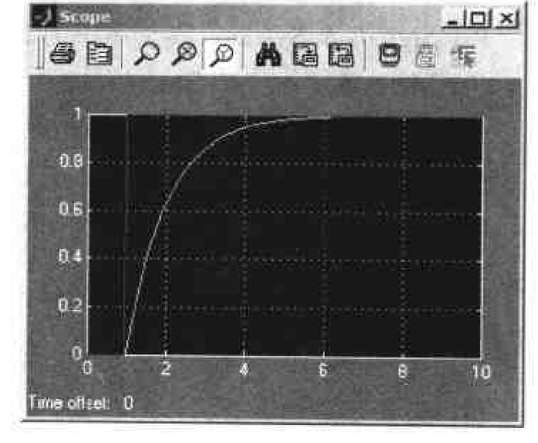
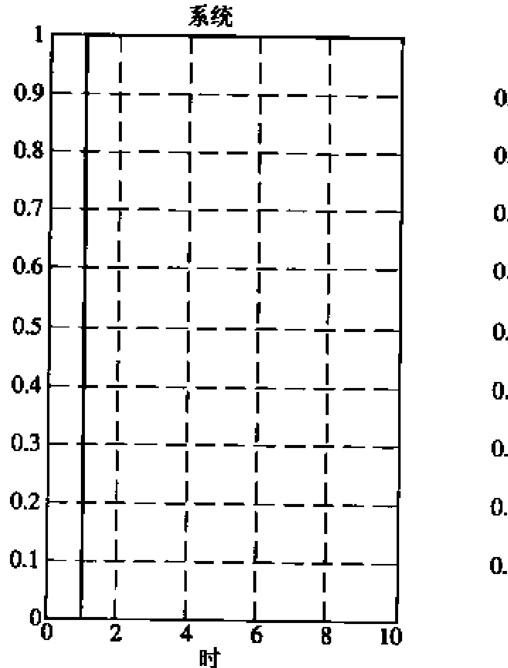


图 4.46　系统仿真结果

注 5：结果输出到 MATLAB 工作空间之中。

使用如下的命令绘制输出曲线，结果如图 4.47 所示。

```
>>subplot(1,2,1),plot(tout,simout(:,2)), grid;        % 绘制输入信号
>> subplot(1,2,2),plot(tout,simout(:,1)), grid;       % 绘制系统仿真输出信号
```



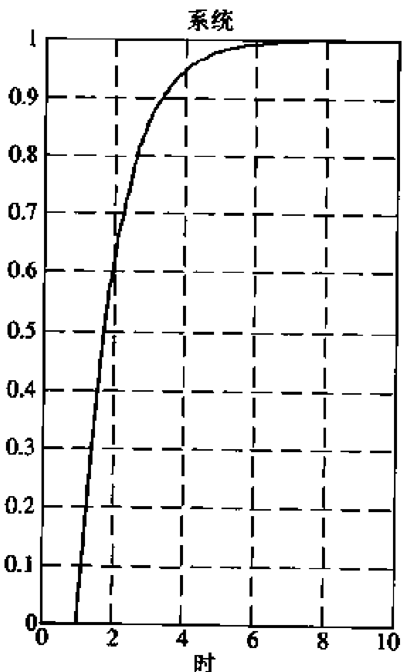


图 4.47　系统仿真输出曲线绘制

第5章　动态系统的 Simulink 仿真

内容概要

- 高级 Scope 使用分析
- 离散系统的仿真分析
- 连续系统的仿真分析
- 混合系统设计分析
- Simulink 的调试技术

在对实际的动态系统进行仿真分析时，往往需要对系统的仿真过程进行各种设置与控制，以达到特定的目的。Simulink 作为一个具有友好用户界面的系统级仿真平台，通过它的图形仿真环境，可以对动态系统的仿真进行各种设置与控制，从而使用户快速完成系统设计的任务。本章将在第 4 章的基础之上，详细介绍各种动态系统（离散系统、连续系统、混合系统）的 Simulink 仿真技术，并对系统仿真参数的设置作全面的解释与说明。

由于本书的主要目的是介绍 Simulink 在动态系统仿真中的应用，而且本书的读者一般对动态系统都有比较深入的了解，因此本章内容采用如下的组织方式：通过对具体的动态系统进行仿真分析，来全面介绍动态系统的 Simulink 仿真技术。

5.1　简单系统的仿真分析

第 3 章对各种动态系统进行了简单的介绍，其中简单系统是指系统方程中不含有状态变量的系统。因此在介绍动态系统的 Simulink 仿真技术时，将首先介绍简单系统的仿真技术。对于下述的简单系统：

$$y(t)=\begin{cases}2u(t), & t>25\\ 10u(t), & t\leqslant 25\end{cases}$$

其中 $u(t)$ 为系统输入，$y(t)$ 为系统输出。下面将建立此简单系统的模型并进行仿真分析。

5.1.1　建立系统模型

首先根据系统的数学描述选择合适的 Simulink 系统模块，然后按照第 4 章中的方法建立此简单系统的系统模型。这里所使用的系统模块主要有：

(1) Sources 模块库中的 Sine Wave 模块：用来作为系统的输入信号。

(2) Math 模块库中的 Relational Operator 模块：用来实现系统中的时间逻辑关系。

(3) Sources 模块库中的 Clock 模块：用来表示系统运行时间。

(4) Nonlinear 模块库中的 Switch 模块：用来实现系统的输出选择。

(5) Math 模块库中的 Gain 模块：用来实现系统中的信号增益。

图 5.1 所示为此简单系统的系统模型。

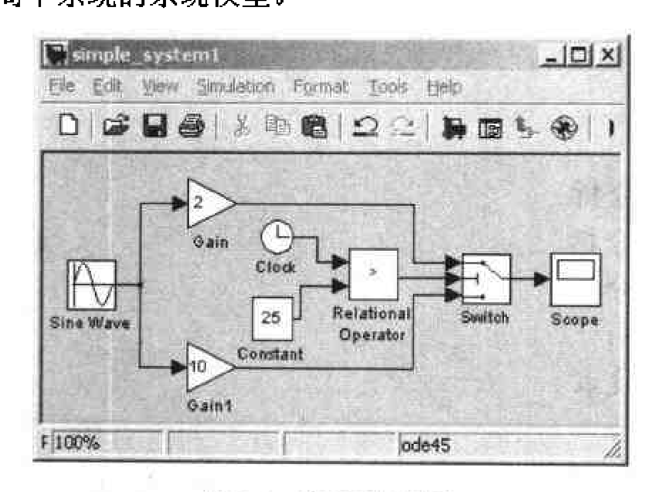


图 5.1 简单系统模型

5.1.2 系统模块参数设置

在完成系统模型的建立之后，需要对系统中各模块的参数进行合理的设置。这里采用的模块参数设置如下所述：

(1) Sine Wave 模块：采用 Simulink 默认的参数设置，即单位幅值、单位频率的正弦信号。

(2) Relational Operator 模块：其参数设置为“>”，如图 5.2 所示。

(3) Clock 模块：采用默认参数设置，如图 5.3 所示。

(4) Switch 模块：设定 Switch 模块的 Threshold 值为 0.5（其实只要大于 0 小于 1 即可，因为 Switch 模块在输入端口 2 的输入大于或等于给定的阈值 Threshold 时，模块输出为第一端口的输入，否则为第三端口的输入），从而实现此系统的输出随仿真时间进行正确的切换，如图 5.4 所示。

(5) Gain 模块：其参数设置如图 5.1 系统模型中所示，这里不再赘述。

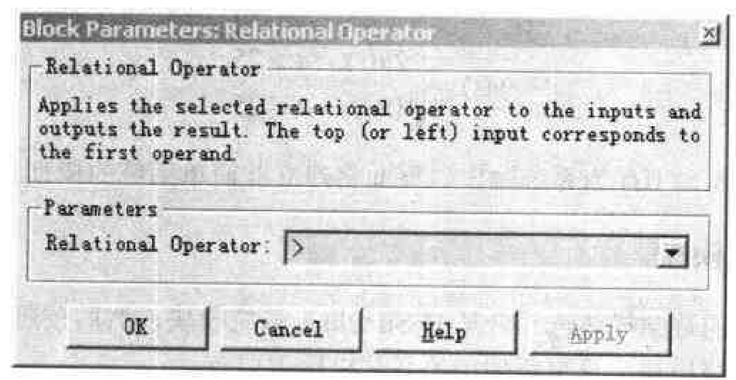


图 5.2 Relational Operator 模块参数设置

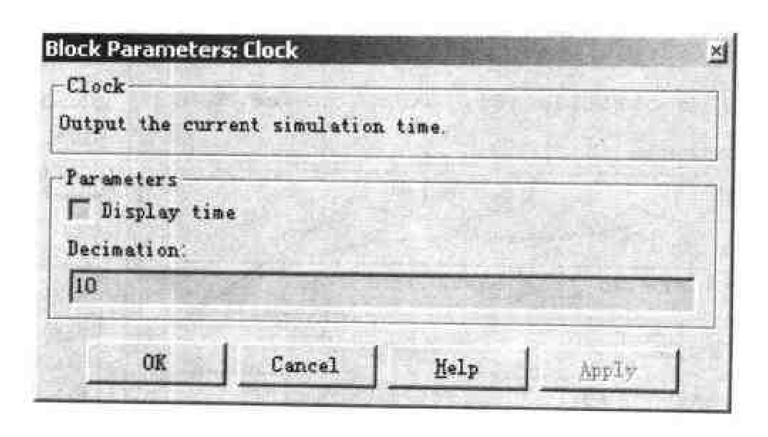


图 5.3　Clock 模块参数设置

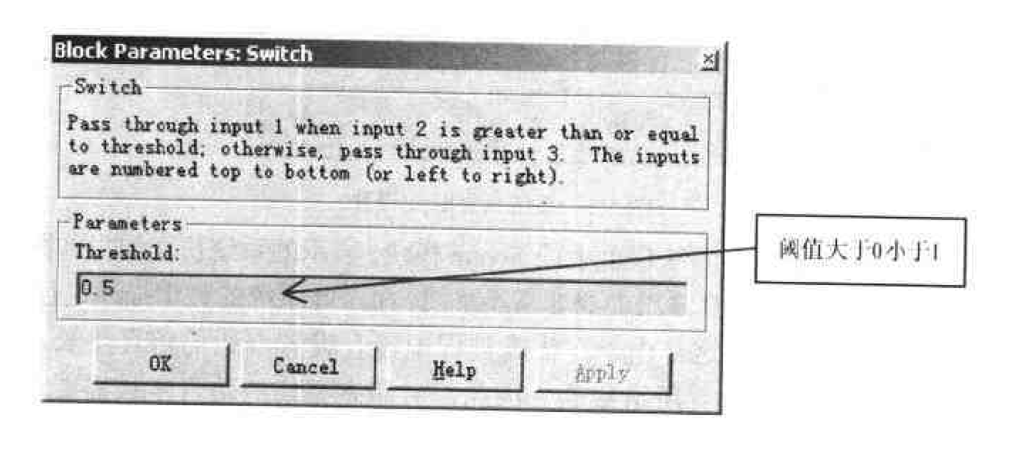


图 5.4　Switch 模块参数设置

5.1.3　系统仿真参数设置及仿真分析

在对系统模型中各个模块进行正确而合适的参数设置之后，便需要对系统仿真参数进行必要的设置以开始仿真。

在缺省情况下，Simulink 默认的仿真起始时间为 0 s，仿真结束时间为 10 s。对于此简单系统，当时间大于 25 s 时系统输出才开始转换，因此需要设置合适的仿真时间。设置仿真时间的方法为：选择菜单 Simulation 中的 Simulation Parameters（或使用快捷键 Ctrl+E），打开仿真参数设置对话框，在 Solver 选项卡中设置系统仿真时间区间。设置系统仿真起始时间为 0 s，结束时间为 100 s，如图 5.5 所示。

在仿真参数设置对话框的 Solver（求解器）选项卡中，可以对系统仿真的求解器进行设置与控制，如求解器类型、求解方法、仿真步长以及误差控制等等。

在系统模块参数与系统仿真参数设置完毕之后，用户便可开始系统仿真了。运行仿真的方法有如下几种：

(1) 选择菜单 Simulation 中的 Start Simulation。

(2) 使用系统组合热键 Ctrl+T。

(3) 使用模型编辑器工具栏中的 Play 按钮（即黑色三角形）。

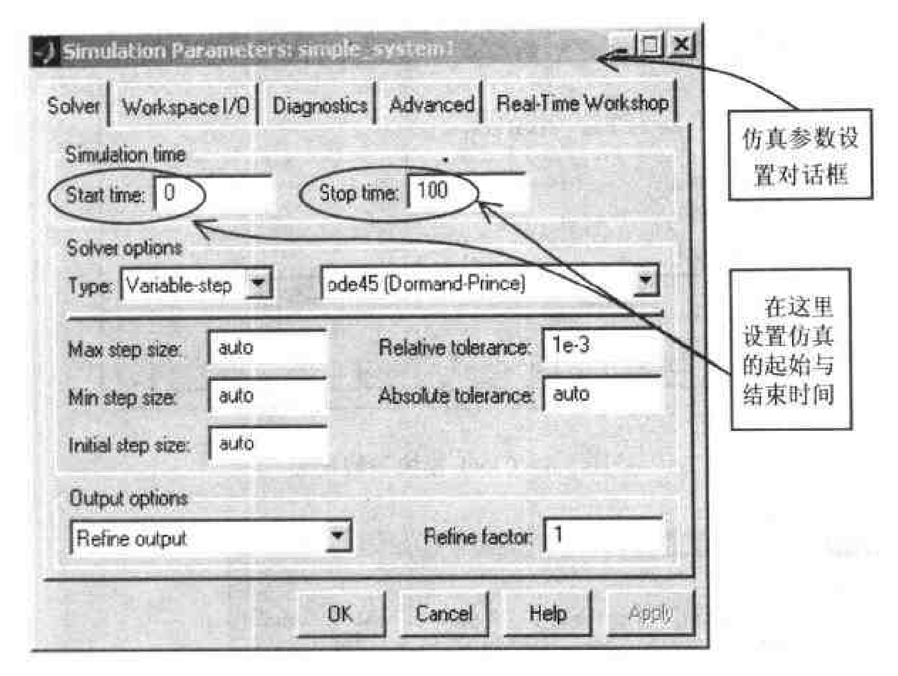


图 5.5 系统仿真时间设置

当系统仿真结束后，双击系统模型中的 Scope 模块，显示的系统仿真结果如图 5.6 所示。从图 5.6 中可以看出，系统仿真输出曲线非常不平滑；而对此系统的数学描述进行分析可知，系统输出应该为光滑曲线。这是由于在仿真过程中没有设置合适的仿真步长，而是使用 Simulink 的默认仿真步长设置所造成的。因此，对动态系统的仿真步长需要进行合适的设置。

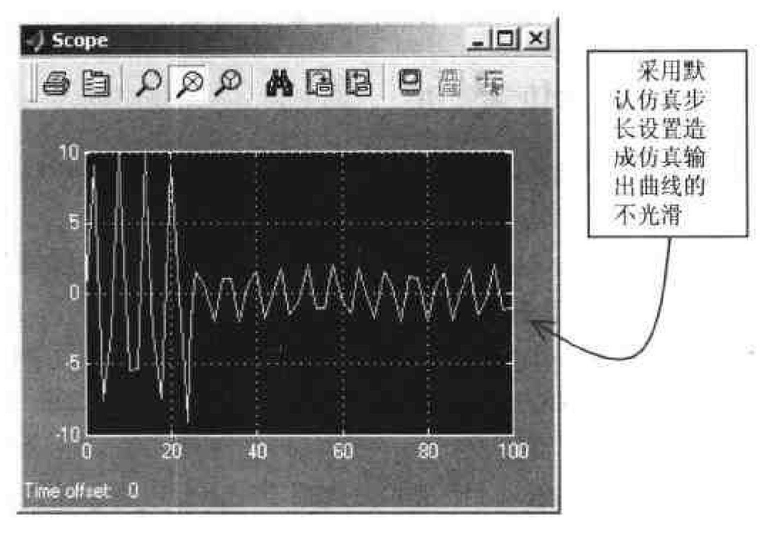


图 5.6 系统仿真结果输出曲线

5.1.4 仿真步长设置

仿真参数的选择对仿真结果有很大的影响。对于简单系统，由于系统中并不存在状态变量，因此每一次计算都应该是准确的（不考虑数据截断误差）。在使用 Simulink 对简单系统进行仿真时，影响仿真结果输出的因素有仿真起始时间、结束时间和仿真步长。对于简单系统仿真来说，不管采用何种求解器，Simulink 总是在仿真过程中选用最大的仿真步长。

如果仿真时间区间较长，而且最大步长设置采用默认取值 auto，则会导致系统在仿真时使用大的步长，因为 Simulink 的仿真步长是通过下式得到的：

$$h=\frac{t_{final}-t_{start}}{50}$$

其中 t_{final} 表示系统仿真的结束时间，而 t_{start} 表示系统仿真的开始时间。在此简单系统中，系统仿真开始时刻为 0 s，结束时刻为 100 s，故仿真步长为 2，从而造成了系统仿真输出曲线很不光滑。用户可以对仿真参数对话框的 Solver 选项卡中的 Max step size（最大步长）进行适当的设置，强制 Simulink 仿真步长不能超过 Max step size。例如，设置此简单系统的最大仿真步长为 0.1，然后进行仿真，则可以获得光滑的系统仿真输出曲线。图 5.7 所示为最大仿真步长设置，图 5.8 所示为在此仿真步长设置下系统的仿真输出结果。

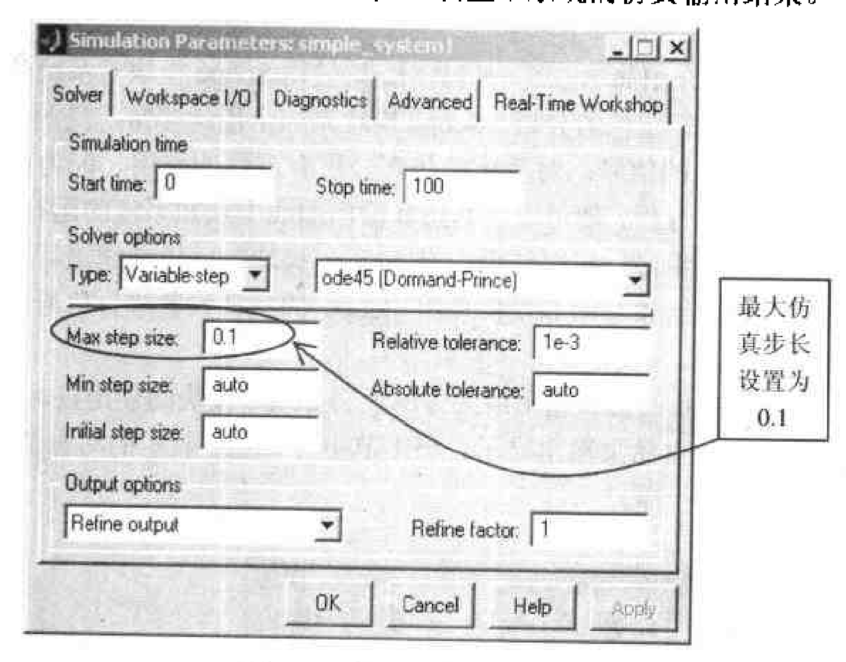


图 5.7　系统最大仿真步长设置

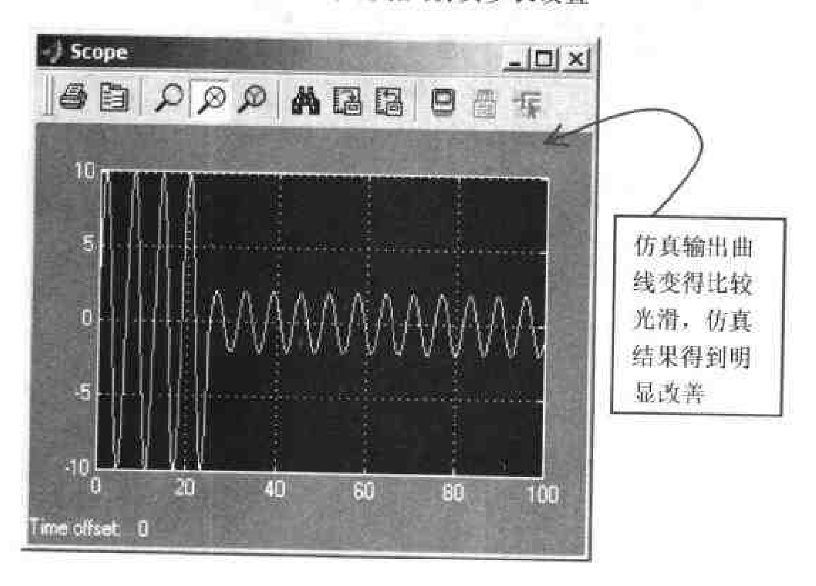


图 5.8　系统最大仿真步长为 0.1 下的仿真输出结果

5.2 Scope 高级使用技术

在对动态系统进行仿真分析时，往往使用 Scope 模块对动态系统的仿真结果或系统模型中指定的信号进行定性的分析。Scope 模块的显示界面与示波器非常类似，用户可以方便地对其进行各种控制以便对输出信号进行观测。此外，用户还可以使用 Sinks 模块库中的 Display 模块输出指定信号的取值。本节将对 Scope 模块、悬浮 Scope 模块以及 Display 模块做详细的介绍。

5.2.1 Scope 模块的使用

在第 4 章及 5.1 节中介绍简单系统的仿真时，均使用 Scope 模块对系统仿真输出结果进行显示，以便用户对其进行定性的分析。Scope 模块是一个用途非常广泛的显示模块，它以图形的方式直接显示指定的信号，对系统的仿真分析有重要的作用，因为在很多情况下，无需对输出结果进行定量分析，便可以从其仿真输出曲线中获知系统的运行规律。Scope 模块给用户提供了诸多控制手段，可以使用户对 Scope 模块的输出曲线进行各种控制调整，以便用户观测和分析输出结果，而非仅仅使用 Scope 的默认设置进行信号显示。下面逐一介绍对 Scope 模块的设置。

这里以 5.1 节中简单系统仿真输出结果为例说明 Scope 模块的使用技术。图 5.9 所示为默认设置下此简单系统仿真结果输出显示。（5.1 节中对 Scope 显示的动态范围进行了调整，因此与图 5.9 中显示的不一样。）

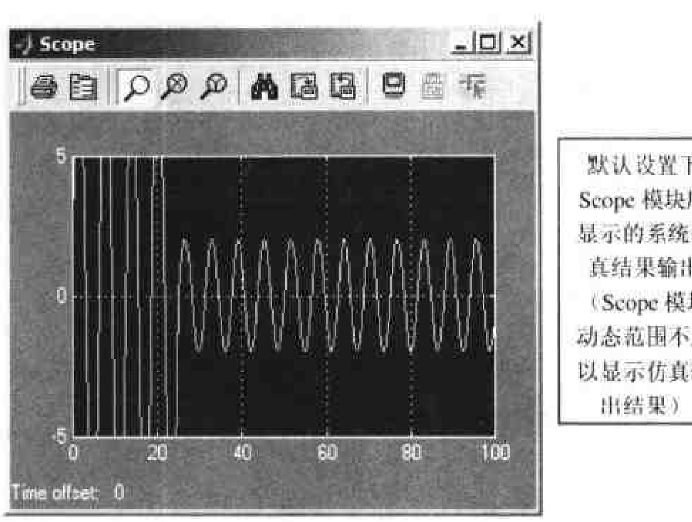


图 5.9 默认设置下 Scope 模块的显示

Scope 模块的工具栏按钮命令如图 5.10 所示。

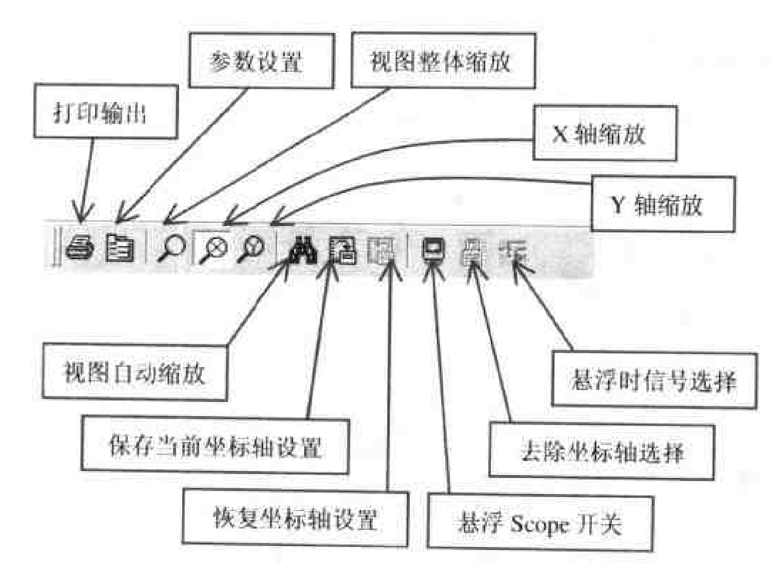


图 5.10 Scope 模块工具栏按钮命令

下面分别对各项功能进行详细介绍。

1) 打印输出（Print）

将系统仿真结果的输出信号打印出来。

2) 视图自动缩放（Autoscale）

Simulink 自动调整显示范围以匹配系统仿真输出信号的动态范围。在图 5.9 中采用默认设置，如果自动缩放视图，则可以获得更好的显示效果，如图 5.11 所示。

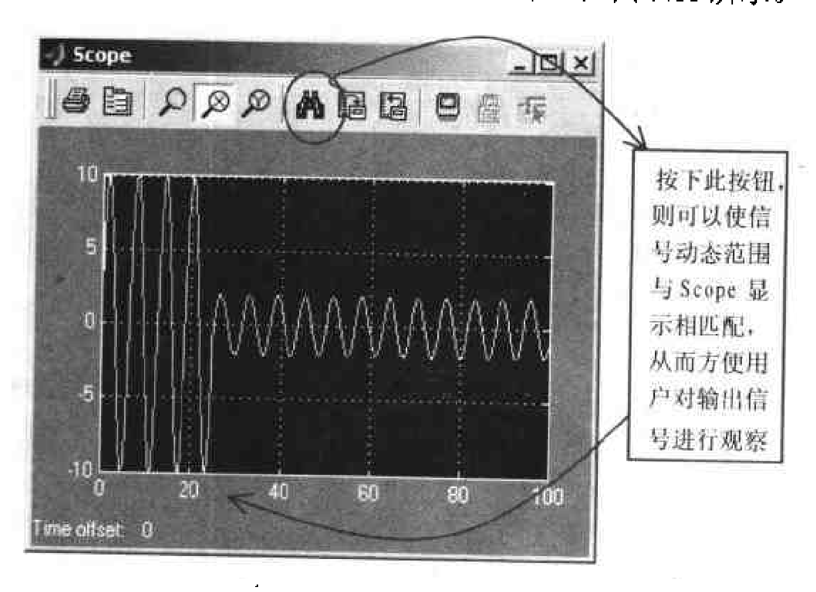


图 5.11 视图自动缩放

3) X 轴缩放、Y 轴缩放以及视图整体缩放

对信号的指定范围进行缩放，可以分别对 X 坐标轴、Y 坐标轴或同时对 X、Y 坐标轴（即整体视图）的信号显示作缩放，以满足用户对信号做局部观察的需要。首先单击缩放按钮，然后选择需要观察的信号范围即可，如图 5.12 所示。如果用户需要缩小视图，单击鼠标右键，选择弹出菜单的 Zoom out 即可。

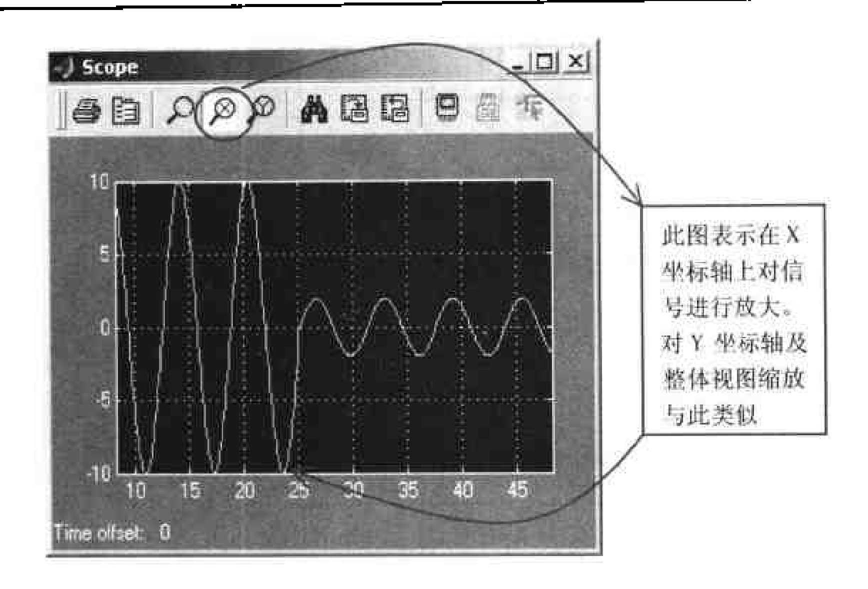


图 5.12　视图缩放

4) 保存与恢复坐标轴设置

在使用 Scope 模块观测输出信号时，用户可以保存坐标轴设置。这样，当信号的视图发生改变后，单击恢复坐标轴设置可以恢复以前保存的坐标轴设置，如图 5.13 所示。

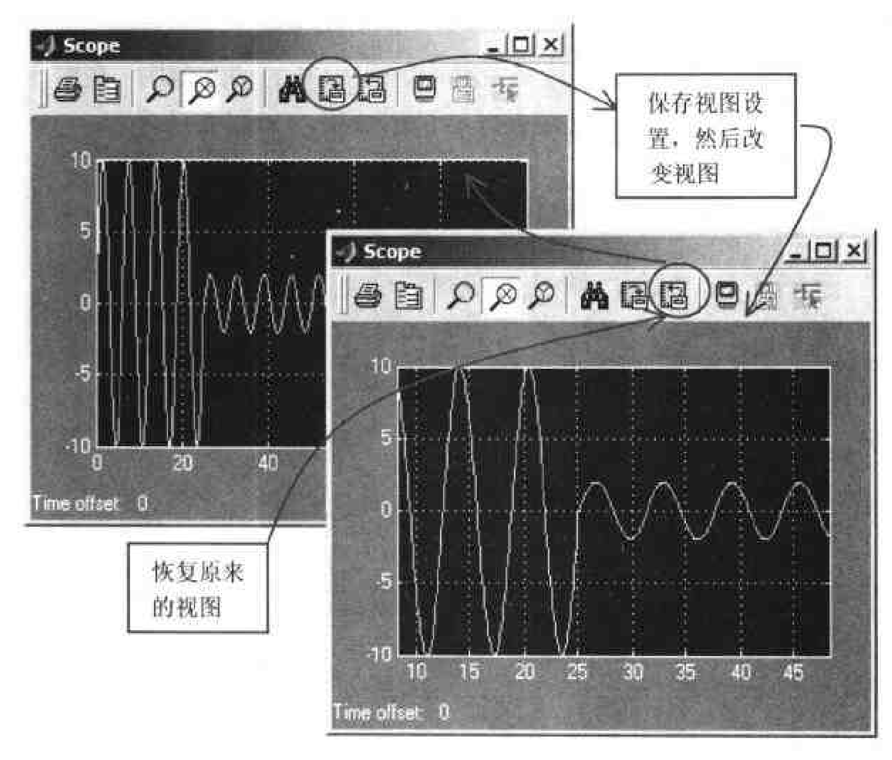


图 5.13　保存与恢复视图设置

5) Scope 的参数设置

使用 Scope 模块的参数设置选项卡，能够对系统仿真输出结果的显示进行更多的控制，而不仅仅是上述的简单控制。图 5.14、5.15 所示分别为 Scope 模块参数设置选项卡中的 General 选项卡与 Data History 选项卡。

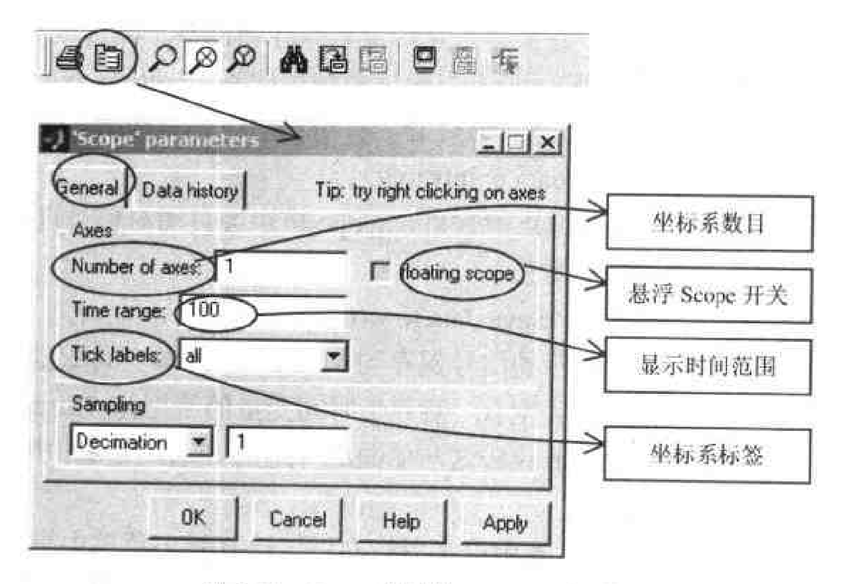


图 5.14　Scope 模块的 General 选项卡

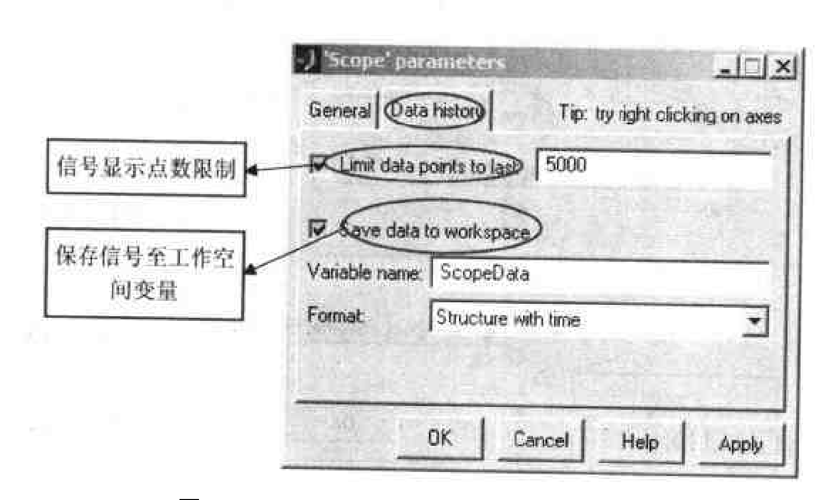


图 5.15　Scope 模块的 Data History 选项卡

下面简单介绍一下各选项卡的功能与使用。

(1) 坐标系数目（Number of axes）。

功能描述：在一个 Scope 输出模块中使用多个坐标系窗口的同时输出多个信号，在默认设置下，Scope 模块仅显示一个坐标系窗口。

(2) 悬浮 Scope 开关（Floating scope）。

功能描述：将 Scope 模块切换为悬浮 Scope 模块。悬浮 Scope 模块将在 5.2.3 节中进行介绍。

(3) 显示时间范围（Time range）。

功能描述：设置信号显示的时间范围。注意：信号显示的时间范围与系统仿真时间范围并不等同，并且坐标系所示的时间范围并非为绝对时间，而是指相对时间范围，坐标系的左下角的时间偏移(Time offset)给出了时间的起始偏移量(即显示时间范围的起始时刻)。

(4) 坐标系标签（Tick labels）。

功能描述：确定 Scope 模块中各坐标系是否带有坐标轴标签。此选项提供了三种选择：全部坐标系都使用坐标轴标签（all）、最下方坐标系使用标签（bottom axis only）以及都不使用标签（none）。用户最好使用标签，这有利于对信号的观察与理解。

(5) 信号显示点数限制（Limit data points to last）。

功能描述：限制信号显示的数据点的数目，Scope 模块会自动对信号进行截取以显示信号的最后 n 个点（这里 n 为设置的数值）。

(6) 保存信号至工作空间变量（Save data to workspace）。

功能描述：将由 Scope 模块显示的信号保存到 MATLAB 工作空间变量中，以便于对信号进行更多的定量分析。数据保存类型有三种：带时间变量的结构体（Structure with time）、结构体（Structure）以及数组变量（Array）。这与前面所介绍的 Sinks 模块库中的 To workspace 模块类似。

此外，在 Scope 模块中的坐标系中单击鼠标右键，选择弹出菜单中坐标系属性设置命令（axes properties），将弹出图 5.16 所示的坐标系属性设置对话框。用户可以对 Scope 模块的坐标系标题与显示信号范围进行合适的设置，以满足仿真输出结果显示的需要。

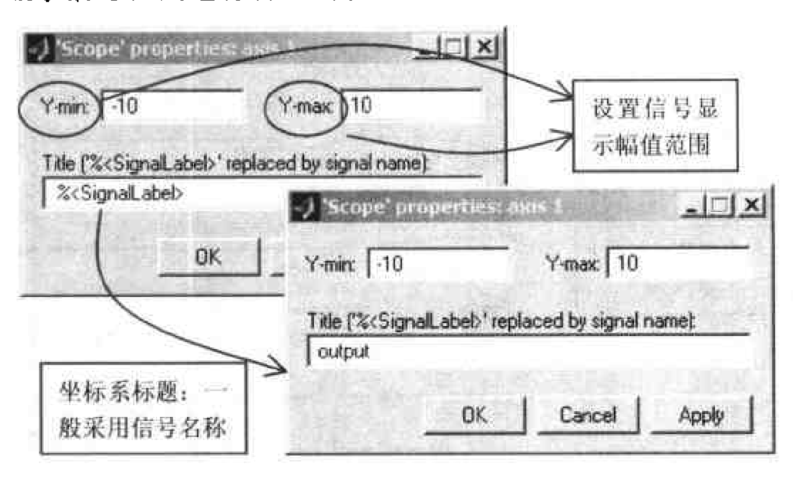


图 5.16 坐标系属性设置对话框

5.2.2 Display 模块的使用

在某些情况下，用户需要观察或动态显示某个信号的数值结果时，可以选用 Display 模块，它既可以显示单个信号，也可以显示向量信号或矩阵信号（帧信号）。当信号的显示范围超出了 Display 模块的边界，会在 Display 模块的右下角出现一个向下的三角，表示还有信号的值没被显示出来，这时用户只需用鼠标拉大 Display 模块的显示面板即可。

5.2.3 悬浮 Scope 模块

在系统仿真分析中，用户往往需要对多个输出信号进行观察分析。如果将每一个信号都与一个 Scope 模块相连接，则系统模型中必定会存在多个 Scope 模块，使得系统模型不够简练，而且难以对不同 Scope 模块中显示的信号进行直观的比较。Sinks 模块库中 Floating Scope 模块（悬浮 Scope 模块）可以很好地解决这一问题。

悬浮模块具有如下的特征：模块没有任何输入与输出端口，不需要和任何信号线连接。悬浮 Scope 模块可以在仿真过程中显示任何选定的信号，而无需修改系统模型。悬浮 Scope

模块与普通 Scope 模块的区别在于：悬浮 Scope 模块可以选定所要显示的信号，而普通 Scope 模块只能显示与之相连的信号。另外，悬浮 Scope 模块可以通过坐标系周围的蓝色框来标志。Display 模块也可以设置为悬浮模式，只需选中它的参数对话框的 Floating display 检查框即可。

1. 悬浮 Scope 模块的使用方法

使用悬浮 Scope 模块的方法有如下两种：

(1) 直接将 Sinks 模块库中的 Floating Scope 模块拖动到指定的系统模型之中。然后选择需要显示的信号并进行适当的设置，最后进行系统仿真并显示系统中指定的信号。

(2) 设置普通的 Scope 模块为 Floating Scope 模块。用户只需选择图 5.14 中所示的悬浮 Scope 开关即可。其后的操作与（1）一致。

例如，对于图 5.17 所示的动态系统模型，使用 Floating Scope 与悬浮 Display 模块显示指定的信号。

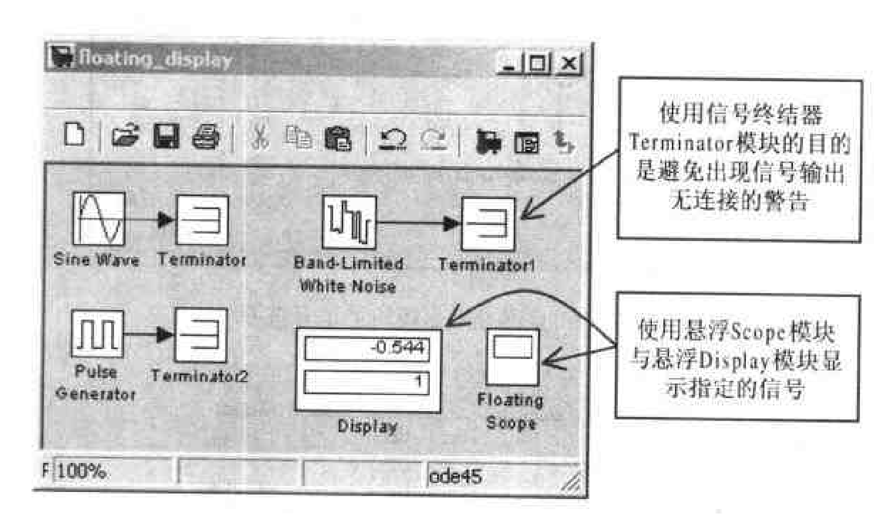


图 5.17　使用悬浮 Scope 模块与悬浮 Display 模块

说明：在此系统模型中的三个信号，正弦信号、方波信号与白噪声信号并不用来驱动其它的模块，仅仅是为了说明如何使用悬浮 Scope 模块显示指定的信号。

2. 使用悬浮 Scope 模块显示信号的参数设置

对于图 5.17 所示的系统模型，要使用悬浮 Scope 模块显示指定的信号，必须进行正确的设置。

1) 设置需要显示的信号

使用悬浮 Scope 模块的信号选择器选择需要显示的信号：首先打开信号选择器对话框，然后在可显示信号列表中选择需要显示的信号，这里选择显示正弦信号与方波信号。信号选择如图 5.18 所示。

2) 设置信号存储缓冲区与全局变量

在缺省情况下，Simulink 重复使用存储信号的缓存区。也就是说，Simulink 信号都是局部变量。使用悬浮 Scope 模块显示指定信号，由于信号与模块之间没有实际的连接，因此局部变量不再适用。故用户应当避免 Simulink 对变量的缓存区重复使用，需要对其进行正确设置。用户可以关闭 Simulink 仿真参数对话框（Simulation Parameters）中 Advanced 选项卡下的 Signal storage reuse 功能（状态设置为 off）；或者将要显示的信号声明为 Simulink

全局变量：选择信号，使用 Edit 菜单下的信号属性设置（Signal Properties）对话框，框选 Signal monitoring and code generation options 中的 Simulink Global (Test Point)即可。信号存储缓冲区设置与全局变量设置如图 5.19 所示。

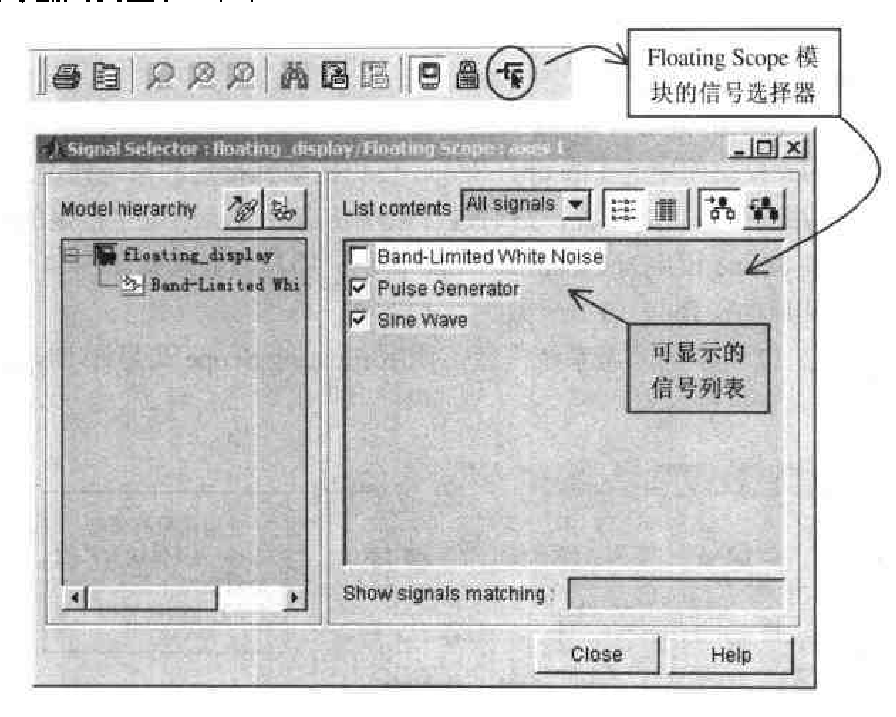


图 5.18　悬浮 Scope 模块的信号选择

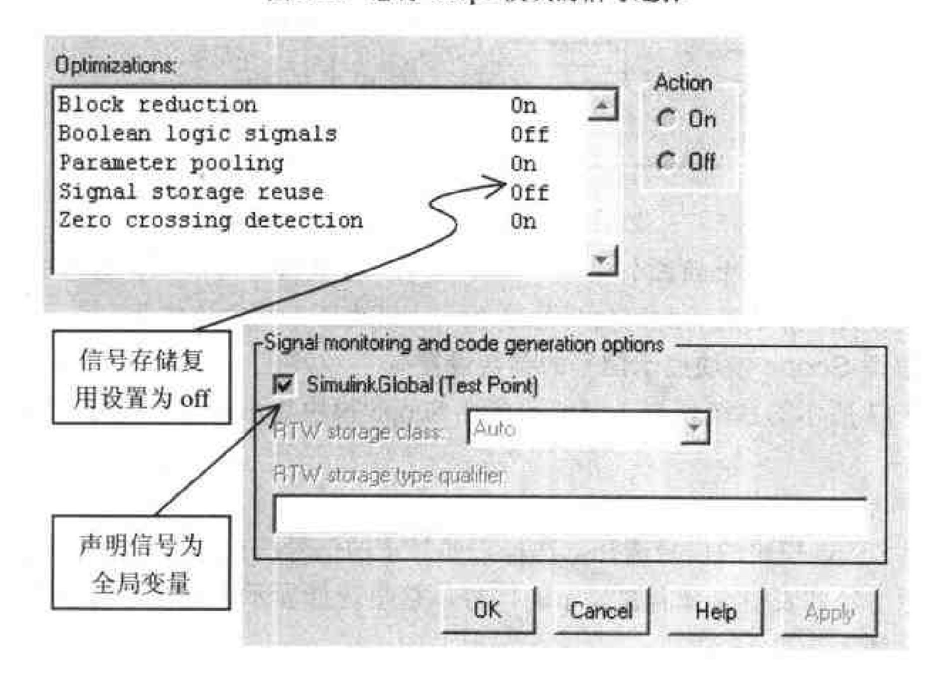


图 5.19　信号存储缓冲区设置

3. 运行系统仿真

在相应的参数设置完成之后，运行系统仿真，系统仿真输出结果如图 5.20 所示。从图中可以看出，正弦信号与方波信号被正确显示出来。

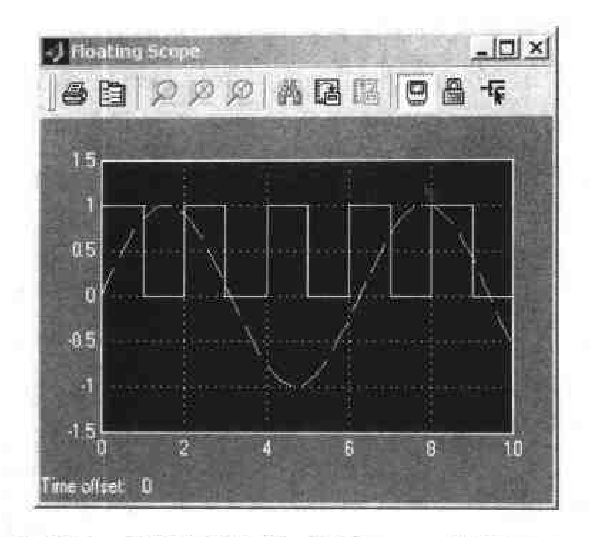


图 5.20　系统仿真结果（悬浮 Scope 模块输出）

5.3　离散系统的仿真分析

5.1 节介绍了简单系统的仿真技术，并对系统仿真步长的设置做了较详细的说明。使用 Simulink 对动态系统进行仿真分析时，需要设置各种仿真参数。本节将对动态离散系统仿真技术进行详细的介绍，并以人口动态变化的非线性离散系统为例继续介绍动态系统仿真参数的设置。

5.3.1　人口变化系统的数学模型

这是一个简单的人口变化模型。在此模型中，设某一年的人口数目为 $p(n)$，其中 n 表示年份，它与上一年的人口 $p(n-1)$ 、人口繁殖速率 r 以及新增资源所能满足的个体数目 K 之间的动力学方程由如下的差分方程所描述：

$$p(n)=rp(n-1)\left[1-\frac{p(n-1)}{K}\right]$$

从此差分方程中可以看出，此人口变化系统为一非线性离散系统。如果设人口初始值 $p(0)=100\,000$ ，人口繁殖速率 $r=1.05$ ，新增资源所能满足的个体数目 $K=1\,000\,000$ ，要求建立此人口动态变化系统的系统模型，并分析人口数目在 0 至 100 年之间的变化趋势。

5.3.2　建立人口变化系统的模型

在建立此人口变化的非线性离散系统模型之前，首先对离散系统模块库（Discrete 模块库）中比较常用的模块作简单的介绍。

(1) Unit Delay 模块：其主要功能是将输入信号延迟一个采样时间，它是离散系统的差分方程描述以及离散系统仿真的基础。在仿真时只要设置延迟模块的初始值便可计算系统输出。

(2) Zero-Order Hold 模块：其主要功能是对信号进行零阶保持。

使用 Simulink 对离散系统进行仿真时，单位延迟是由 Discrete 模块库中的 Unit Delay

模块来完成的。对于人口变化系统模型而言，需要将 $p(n)$ 作为 Unit Delay 模块的输入以得到 $p(n-1)$，然后按照系统的差分方程来建立人口变化系统的模型。

由于此系统的结构比较简单，所以这里直接给出系统的模型框图，如图 5.21 所示。

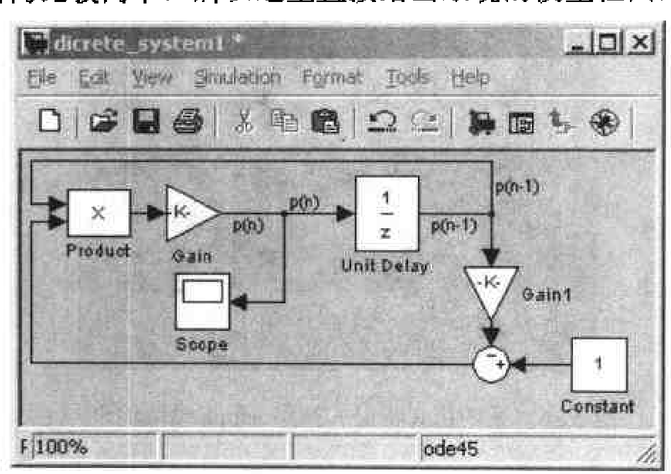


图 5.21 人口变化系统模型

注意：此人口变化系统模型中没有输入信号，只需给出人口的初始值便可进行仿真；另外，增益模块 Gain 表示人口繁殖速率 r，而模块 Gain1 表示新增资源所能满足的个体数目 K 的倒数（即 $1/K$）。

5.3.3 系统模块参数设置

系统模型建立之后，首先需要按照系统的要求设置各个模块的参数，如下所述：

(1) 增益模块 Gain 表示人口繁殖速率，故取值为 1.05。

(2) 模块 Gain1 表示新增资源所能满足的个体数目的倒数，故取值为 1/100 000。

(3) Unit Delay 模块参数设置。对于离散系统而言，必须正确设置所有离散模块的初始取值，否则系统仿真结果会出现错误。这是因为在不同的初始值下，系统的稳定性会发生变化。单位延迟模块的参数设置如图 5.22 所示。

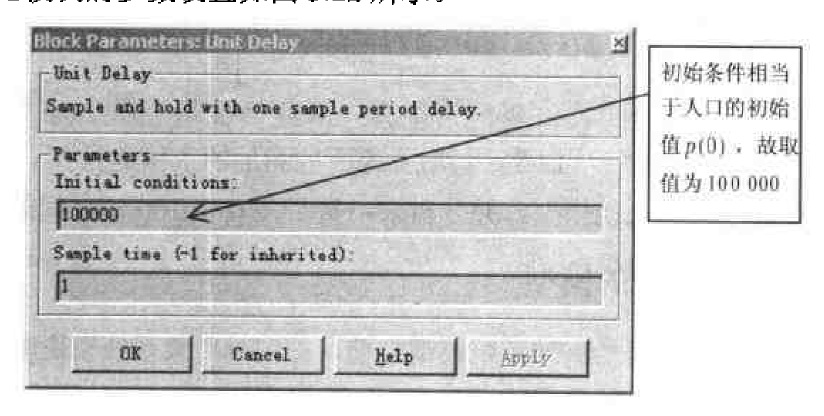


图 5.22 单位延迟模块的参数设置

5.3.4 系统仿真参数设置及仿真分析

在正确设置系统模型中各模块的模块参数之后，需要对系统仿真参数进行设置。下面

介绍离散系统的仿真参数设置，在此之前首先介绍系统仿真的基本原理。这可以使用户加深对离散系统仿真的理解，并且更好的掌握离散系统仿真技术。系统的仿真原理如下所述：Simulink 通过系统模型（框图）与 MATLAB 的求解器之间的交互对话完成离散系统的仿真，Simulink 传递模块参数以及差分（微分）方程给 MATLAB 求解器，而 MATLAB 求解器计算系统模块的输出以更新离散系统状态并确定下一步仿真时间，如图 5.23 所示。

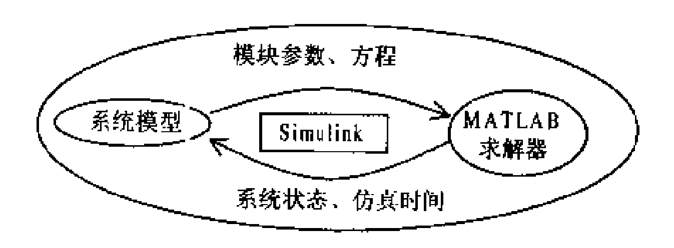


图 5.23　系统的仿真原理

下面设置人口变化系统的仿真参数：

(1) 仿真时间设置：按照系统仿真的要求，设置系统仿真时间范围为 0~100 s。

(2) 离散求解器与仿真步长设置：对离散系统进行仿真需要使用离散求解器。对于离散系统的仿真，无论是采用定步长求解器还是采用变步长求解器，都可以对离散系统进行精确的求解。这里选择定步长求解器对此系统进行仿真分析。至于定步长与变步长的区别将在后面专门进行介绍。

使用 Simulation 菜单中的 Simulation Parameters 设置系统仿真参数，如图 5.24 所示。

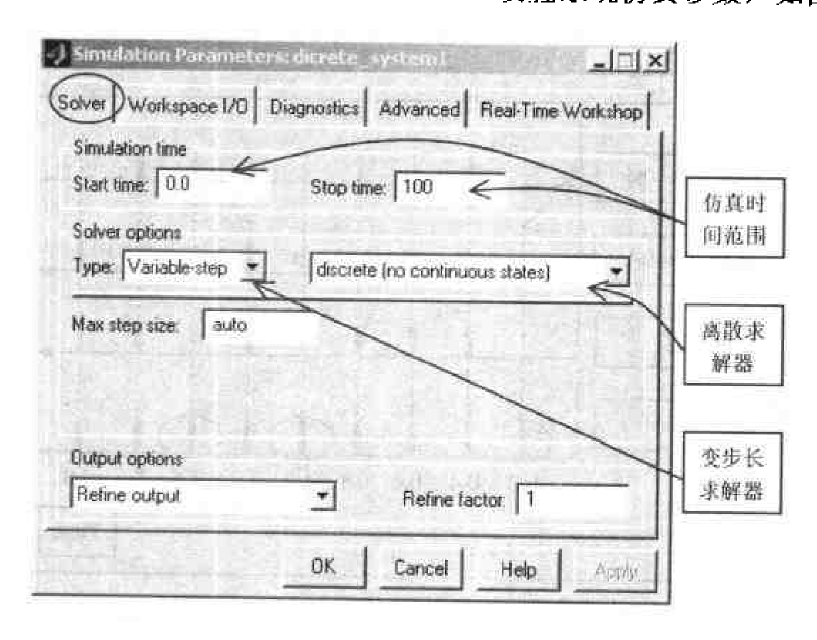


图 5.24　系统仿真参数设置

在对系统中各模块参数以及系统仿真参数进行正确设置之后，运行系统仿真，对人口数目在指定的时间范围之内的变化趋势进行分析。图 5.25 所示为系统仿真输出结果。

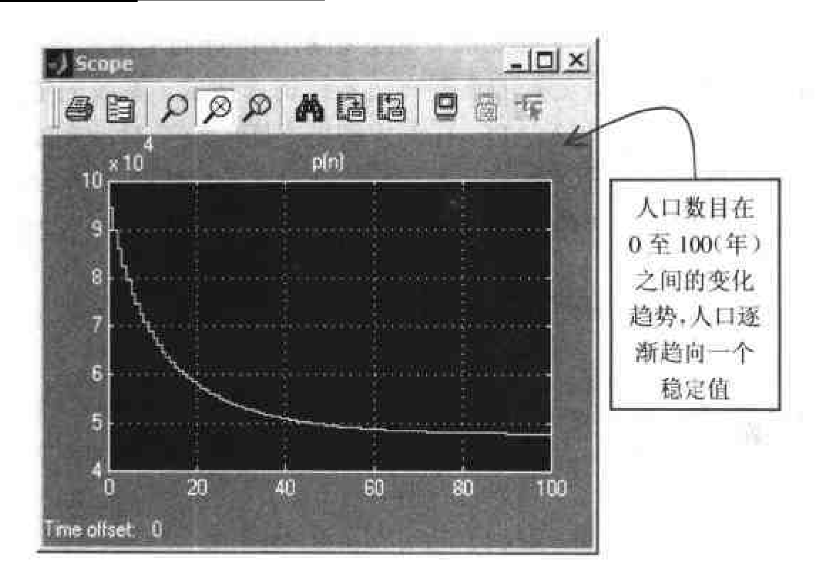


图 5.25 人口变化系统仿真结果

5.3.5 定步长仿真与变步长仿真

对于一个单速率离散系统的仿真，选择定步长求解器对仿真来说已经足够了。但是在对多速率离散系统进行仿真时，采用变步长仿真则具有更多的优势。这里所谓的单速率与多速率离散系统，是指离散系统中各系统模块采样时间是否一致，如果所有模块的采用时间均相同，则此系统为单速率离散系统，否则为多速率离散系统。一般而言，使用变步长求解器对离散系统进行仿真，其效率要优于定步长求解器，如图 5.26 所示。

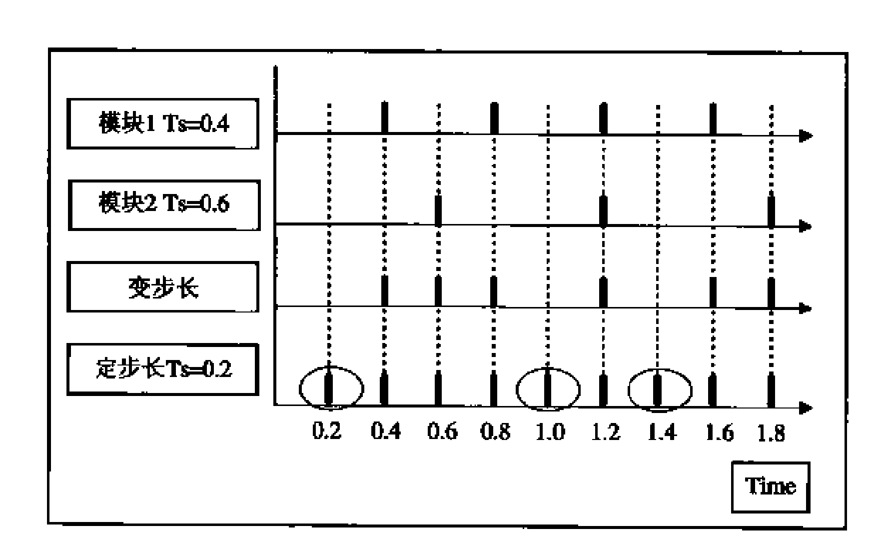


图 5.26 定步长求解器与变步长求解器的仿真时刻对比

图 5.26 表示某一离散系统模型中的某两个系统模块具有不同的采样时间，模块 1 的采样时间为 0.4 s，而模块 2 的采样时间为 0.6 s。图中每个黑色粗实线代表了模块的一次更新，也就是模块的每一次采样。如果使用定步长求解器对此离散系统进行仿真，由于模块之间采样时间不同，因此步长选择要足够小，以匹配所有的采样时刻，于是导致在某些时刻系统进行了不必要的计算（图中由椭圆曲线标志）。如果使用变步长求解器，仿真步长自动调整使其恰好匹配两个模块的更新，从而提高了系统仿真的效率。

注意： 在使用定步长求解器对离散系统进行仿真时，离散系统模块的采样时间必须是用户给定的仿真步长的整数倍，否则系统将报告出错。

5.4 连续系统的仿真分析

5.1 节和 5.2 节分别对简单系统、离散系统的仿真技术做了比较全面的介绍。然而对于实际的动态系统而言，大都是具有连续状态的连续系统。对具有连续状态的连续系统的仿真与不带状态的简单连续系统不同，这是因为简单连续系统中不涉及到输入或输出的导数，而具有连续状态的连续系统中存在着输入或输出的导数。从某种意义上来说，连续系统中的连续状态是一种“记忆元素”，它可以保存系统中的信息并用来直接计算系统的输出，而不需要涉及导数。本节通过对蹦极跳（Bungee Jumping）系统的仿真来介绍连续系统的仿真技术。

5.4.1 蹦极跳系统的数学模型

蹦极跳是一种挑战身体极限的运动，蹦极者系着一根弹力绳从高处的桥梁（或山崖等）向下跳。在下落的过程中，蹦极者几乎处于失重状态。按照牛顿运动规律，自由下落的物体的位置由下式确定：

$$m\ddot{x} = mg - a_1\dot{x} - a_2|\dot{x}|\dot{x}$$

其中 m 为物体的质量，g 为重力加速度，x 为物体的位置，第二项与第三项表示空气的阻力。其中位置 x 的基准为蹦极者开始跳下的位置（即选择桥梁作为位置的起点 $x=0$），低于桥梁的位置为正值，高于桥梁的位置为负值。如果物体系在一个弹性常数为 k 的弹力绳索上，定义绳索下端的初始位置为 0，则其对落体位置的影响为

$$b(x) = \begin{cases} -kx, & x > 0 \\ 0, & x \leqslant 0 \end{cases}$$

因此整个蹦极跳系统的数学描述为

$$m\ddot{x} = mg + bx - a_1\dot{x} - a_2|\dot{x}|\dot{x}$$

从蹦极跳系统的数学描述中可以得知，此系统为一典型的具有连续状态的非线性连续系统。设桥梁距离地面为 50 m，蹦极者的起始位置为绳索的长度 −30 m，即 $x(0) = -30$，蹦极者起始速度为 0，即 $\dot{x}(0) = 0$；其余的参数分别为 k=20，a_2=a_1=1，m=70 kg，g=10 m/s^2。下面将建立蹦极跳系统的仿真模型，并在如上的参数下对系统进行仿真，分析此蹦极跳系统对体重为 70 kg 的蹦极者而言是否安全。

5.4.2 建立蹦极跳系统的 Simulink 仿真模型

与建立离散系统模型类似，在建立蹦极跳系统的模型之前，首先对连续系统模块库 Continuous 中比较常用的模块进行简单的回顾。

(1) 积分器（Integrator）：积分器的主要功能在于对输入的连续信号进行积分运算。积分器是建立连续系统微分方程的基础，也就是建立连续系统模型的基础；同时它也是 Simulink 对具有连续状态的连续系统仿真的基础。在连续系统中通常使用积分器来实现系统中的微分运算，一个积分器模块表示一阶微分，高阶微分则由多个积分器模块串联构成。

注意：必须对所有的积分器设置合适的初始条件（初始状态）。

(2) 微分器（Derivative）：微分器的主要功能在于对输入的连续信号进行微分运算。虽然在连续系统的数学描述中使用连续状态的微分（导数），但是一般不提倡直接使用微分器建立系统模型。一般只有当微分方程中包含系统输入的微分时才使用。

在蹦极跳系统模型中，主要使用的系统模块有：

(1) Continuous 模块库中的 Integrator 模块：用来实现系统中的微分运算。

(2) Functions & Tables 模块库中的 Fcn 模块：用来实现系统中空气阻力的函数关系。

(3) Nonlinear 模块库中的 Switch 模块：用来实现系统中弹力绳索的函数关系。

蹦极跳系统的模型框图如图 5.27 所示。

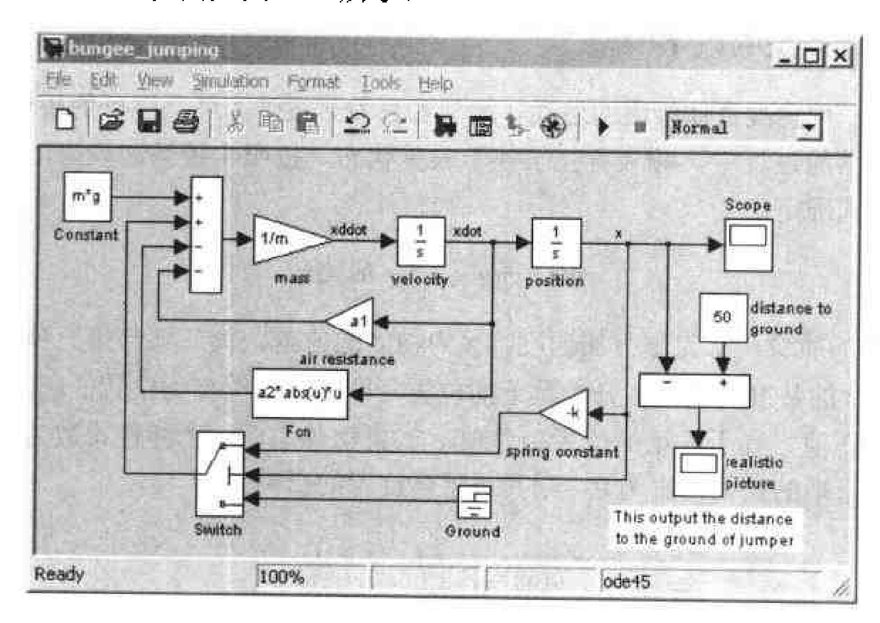


图 5.27 蹦极跳系统模型

在蹦极跳系统模型中使用了两个 Scope 输出模块，上面的 Scope 模块用来显示蹦极者的相对位置，即相对于桥梁的位置；而下面的 Scope 模块用来显示蹦极者的绝对位置，即相对于地面的距离。

5.4.3 系统模块参数设置

在建立蹦极跳系统模型之后，需要设置系统模型中各个模块的参数。这里仅给出积分器模块 velocity 与 position 的参数设置，如图 5.28 所示。

在具有连续状态的连续系统中，千万不能忘记对积分器模块的初始值进行设置。因为在不同的初始值下，系统的动态规律可能大相径庭。至于其它模块的参数都比较简单，这里不再给出。

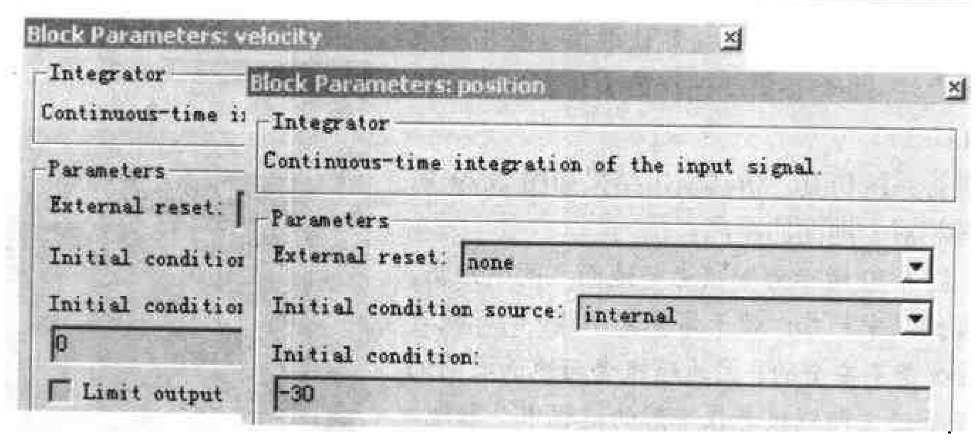


图 5.28　积分器模块 velocity 与 position 的参数设置

注意：这里使用 MATLAB 工作空间中的变量作为系统模块的参数（蹦极者质量 m，重力加速度 g，弹性常数 k，常数 a_1 与 a_2，如图 5.27 所示），因此在系统仿真之前需要对这些变量进行赋值。采用 MATLAB 工作空间变量的主要目的是增加系统的可理解性。

5.4.4　系统仿真参数设置与仿真分析

在对蹦极跳系统模型中各个模块的参数正确设置之后，需要设置系统仿真参数以对此系统进行仿真分析。在系统仿真参数设置之前，首先简单介绍一下 Simulink 的连续求解器。

5.3 节中对离散系统的仿真原理做了简单的介绍，Simulink 通过离散系统模型与 MATLAB 求解器之间的交互完成离散系统的仿真；其实对于任何的动态系统，Simulink 总是通过系统模型与 MATLAB 求解器之间的交互来完成系统仿真。但是 Simulink 的连续求解器与离散求解器有着本质的区别。这是因为，对于离散系统而言，系统仿真分析的基础是差分方程，离散求解器能够对差分方程进行精确求解（不考虑数据截断误差）；而对于具有连续状态的连续系统而言，系统仿真分析的基础是微分方程，而 MATLAB 对微分方程只能进行数值求解以获得近似的结果。因此使用 Simulink 连续求解器对具有连续状态的连续系统进行仿真时，必定存在着一定的误差。

微分方程的不同数值求解方法对应着不同的连续求解器。Simulink 的连续求解器可以使用不同的数值求解方法对连续系统进行求解：

(1) 定步长连续求解器。可以使用如下的方法：ode5, ode4, ode3, ode2, ode1。

(2) 变步长连续求解器。可以使用如下的方法：ode45, ode23, ode113, ode15s, ode23s, ode23t，ode23tb。

定步长求解器使用固定的仿真步长对连续系统进行求解，使用定步长求解器不能对系统中积分误差进行控制；而变步长求解器则能够根据用户指定的积分误差自动调整仿真步长，也就是说，变步长求解器能够对积分求解误差进行控制。另外，在对刚性连续系统（系统中某些状态变化非常剧烈的系统）进行仿真分析时，应该采用刚性求解算法 ode15s、ode23s、ode23t 与 ode23tb 等进行求解。

使用 Simulation 菜单下的 Simulation Parameters 打开仿真参数设置对话框，对蹦极跳系统的仿真参数设置如下：

(1) 系统仿真时间范围为 0~100 s。

(2) 其它仿真参数采用系统默认取值（变步长求解器、求解算法 ode45、自动选择最大仿真步长、相对误差为 1e−3）。

然后进行系统仿真。仿真输出结果如图 5.29 所示（蹦极者相对于地面的距离）。

注意：图 5.29 所示为蹦极者与地面之间的距离。从系统仿真结果中可知：对于体重为 70 kg 的蹦极者来说，此系统是不安全的，因为蹦极者与地面之间的距离出现了负值（即蹦极者在下落的过程中会触地，而安全的蹦极跳系统要求二者之间的距离应该大于 0）。因此，必须使用弹性常数较大的弹性绳索，才能保证蹦极者的安全。当然，在蹦极者触地的情况下，系统的动态方程会发生改变，系统输出结果也将发生变化。图 5.29 给出的仿真结果并没有考虑这一点（假设蹦极者距离地面足够大，不会触地）。

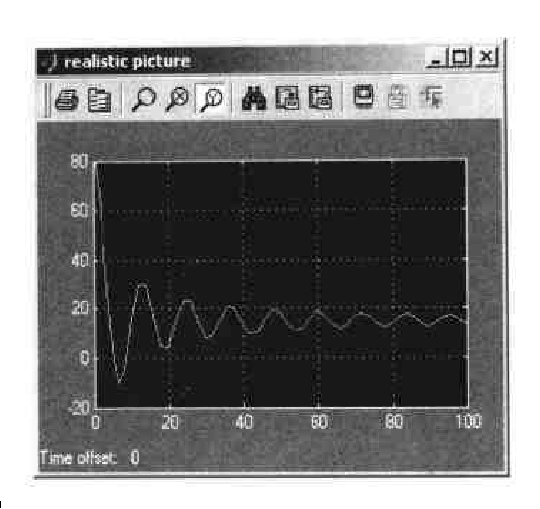


图 5.29 蹦极跳系统的仿真结果

5.4.5 仿真精度控制

细心的读者一定会发现，在图 5.29 所示蹦极跳系统的仿真结果中，仿真曲线的波峰与波谷处曲线很不光滑。而从蹦极跳系统的数学方程中分析可知，系统的输出曲线应该是光滑曲线。造成这一结果的主要原因是：对此系统仿真来说，连续求解器的默认积分误差取值偏大。因此，只有设置合适的积分误差限，才能获得更好的仿真结果。

变步长连续求解器可以根据积分误差对仿真步长进行自动调整。因此，用户可以设置合适的积分误差上限，以对系统仿真步长进行控制，从而获得更好的仿真结果。积分误差分为如下两种：

(1) 绝对误差：积分误差的绝对值。

(2) 相对误差：绝对误差除以状态的值。

在仿真参数设置对话框中，用户可以对积分绝对误差与相对误差进行合适的设置。这样，在系统仿真中求解器只有满足给定的误差条件时才能够进行下一步计算。一般来说，当状态值较大时，相对误差小于绝对误差，由相对误差控制求解器的运行；而当状态值接近零时，绝对误差小于相对误差，则由绝对误差来控制求解器。

注意：虽然减小积分误差限可以提高系统仿真结果的精度，但是这在一定程度上会影响系统仿真的效率，使系统仿真速度变慢；使用较大的积分误差限或使用定步长求解器可以加快系统的仿真速度，但会使仿真结果的精度降低。故在实际使用中，需要综合考虑系统仿真精度与仿真效率。此外，在使用变步长求解器时，用户需要设置合适的初始仿真步长与最大仿真步长。这是因为对于某些系统而言，系统仿真需要特殊的启动条件，不合适的初始仿真步长有可能导致系统进入不稳定状态。

对蹦极跳系统的积分误差、最大仿真步长与起始仿真步长进行合适的设置，如图 5.30 所示。

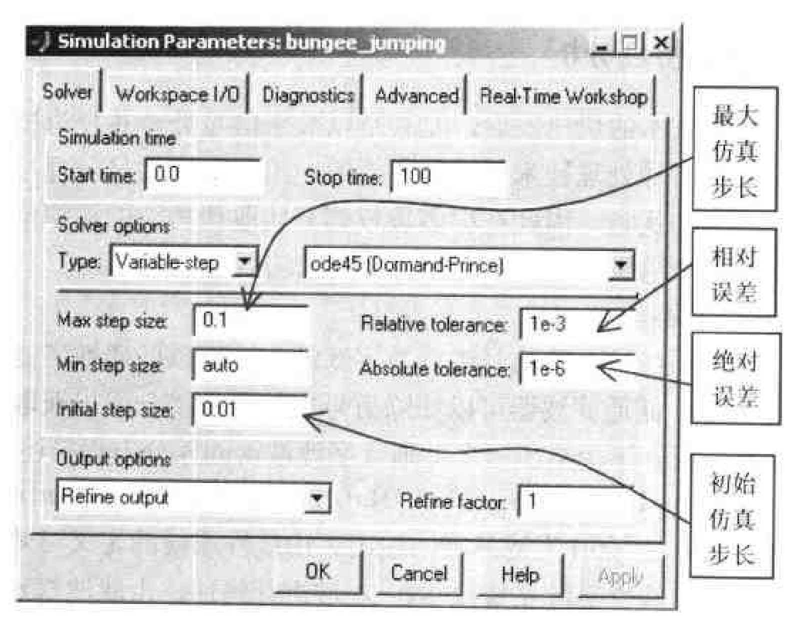


图 5.30 蹦极跳系统的仿真参数设置

然后对蹦极跳系统进行仿真，其仿真结果如图 5.31 所示。从图中可以明显看出，减小系统仿真积分误差可以有效地提高系统的仿真性能，使仿真输出波峰与波谷处的曲线变得比较光滑。

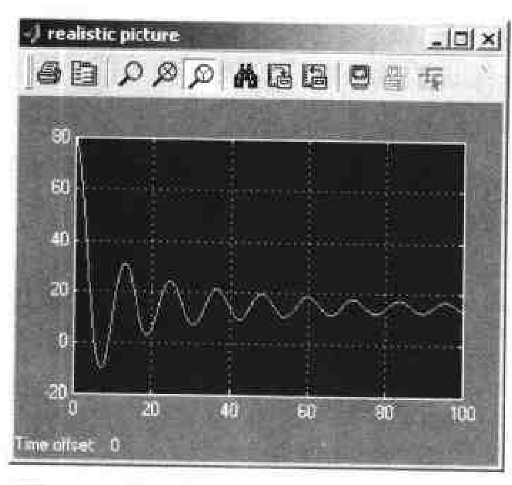


图 5.31 新的仿真参数设置下的仿真结果

5.5 线性系统仿真分析

前面对简单系统、离散系统以及连续系统的 Simulink 仿真技术做了比较全面的介绍。只需掌握这些知识，用户就可以在建立系统模型的基础之上，对动态系统进行仿真分析。在实际的系统中，虽然多数为非线性系统，但是非线性系统的设计、仿真与分析都远比线性系统复杂。因此，在很多情况下，用户总是通过各种方法对非线性系统进行线性化处理，以简化系统的设计、仿真与分析。前面所介绍的离散系统与连续系统均是非线性系统，本节主要介绍线性系统的仿真分析。

5.5.1 线性离散系统仿真分析

随着数字信号处理技术的快速发展，尤其是以数字信号处理芯片为核心的数字系统的设计与使用，使得数字信号处理技术得到了广泛的应用。数字信号处理技术具有诸多模拟信号处理技术所不具备的优点，因此在很多领域都开始取代传统的模拟信号处理。下面以数字滤波器系统为例来介绍线性离散系统的仿真技术。

1. 数字滤波器的数学描述

数字滤波器可以对系统输入的信号进行数字滤波。这里以低通数字滤波器为例说明线性离散系统的仿真技术。低通滤波器可以滤除信号中的高频部分，以获取信号中有用的低频信号，其使用非常广泛。下面给出一个低通数字滤波器的差分方程描述：

$$y(n)-1.6y(n-1)+0.7y(n-2)=0.04u(n)+0.08u(n-1)+0.04u(n-2)$$

其中$u(n)$为滤波器的输入，$y(n)$为滤波器的输出。由线性系统的定义可知，此低通数字滤波器为一线性离散系统。线性离散系统往往在 Z 域进行描述，由滤波器系统的差分方程可获得系统的 Z 变换域描述：

$$\frac{Y(z)}{U(z)}=\frac{0.04+0.08z^{-1}+0.04z^{-2}}{1-1.6z^{-1}+0.7z^{-2}}$$

2. 建立数字滤波器系统模型

这里使用简单的通信系统说明低通数字滤波器的功能。在此系统中，发送方首先使用高频正弦波对一低频锯齿波进行幅度调制，然后在无损信道中传递此幅度调制信号；接收方在接收到幅度调制信号后，首先对其进行解调，然后使用低通数字滤波器对解调后的信号进行滤波以获得低频锯齿波信号。

建立此系统模型所需要的系统模块主要有：

(1) Sources 模块库中的 Sine Wave 模块：用来产生高频载波信号 Carrier 与解调信号 Carrier1。

(2) Sources 模块库中的 Signal Generator 模块：用来产生低频锯齿波信号 sawtooth。

(3) Discrete 模块库中的 Discrete Filter 模块：用来表示数字滤波器。

(4) Math 模块库中的 Product 模块：用来完成低频信号的调制与解调。

此外，使用高频正弦波对低频锯齿波信号进行调制时，所采用的调制方法为双边带抑制载波调制：output=input×carrier，其中 input 为低频信号，carrier 为高频载波信号，output 为幅度调制信号。建立数字滤波器系统模型如图 5.32 所示。

说明：建立线性离散系统时，还经常使用到 Discrete 模块库离散状态空间模块、离散零极点模块以及离散传递函数模块等模块，它们的使用比较简单，这里不作过多介绍。

3. 系统模块参数设置

在数字滤波器系统模型建立之后，需要对模型中各个系统模块进行如下的参数设置：

(1) 正弦载波信号模块 Carrier 的参数设置：频率 Frequency 为 1000 rad/sec，其余设置为默认值。

(2) 信号发生器模块 Signal Generator 参数设置：Wave form 设置为 sawtooth，幅值与频率均设置为 1（默认值）。

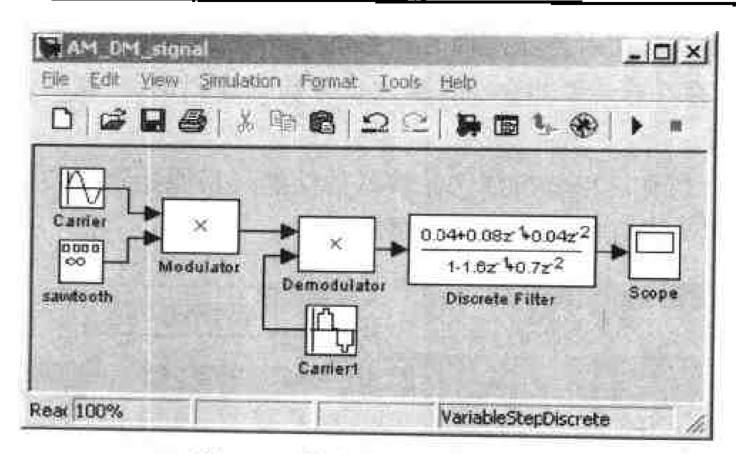


图 5.32　数字滤波器系统模型

(3) 正弦解调信号模块 Carrier1 参数设置：频率为 1000 rad/sec，采样时间 Sample time 为 0.005 s，其余设置为默认值。

(4) 数字滤波器模块 Discrete Filter 参数设置：分子多项式 numerator 为[0.04 0.08 0.04]，分母多项式为[1 -1.6 0.7]，采样时间 Sample time 为 0.005 s。

由于对模块的参数设置非常简单，因此这里不再给出相应的模块参数设置对话框，但是需要说明以下几点：

(1) 信号发生器模块 Signal Generator 可以用来产生多种信号，如方波信号、正弦信号、锯齿波信号及随机信号等，使用时只需选择相应的信号即可。

(2) 解调信号为离散信号，主要是为了使数字滤波器的输入信号为一数字信号。

(3) 数字滤波器的采样时间一般应与解调信号的采样时间保持一致。

4. 系统仿真参数设置与仿真分析

在系统模块参数设置完毕之后，应设置系统仿真参数，最后进行系统的仿真分析。在此使用变步长连续求解器对此系统进行仿真分析。仿真参数设置如图 5.33 所示。

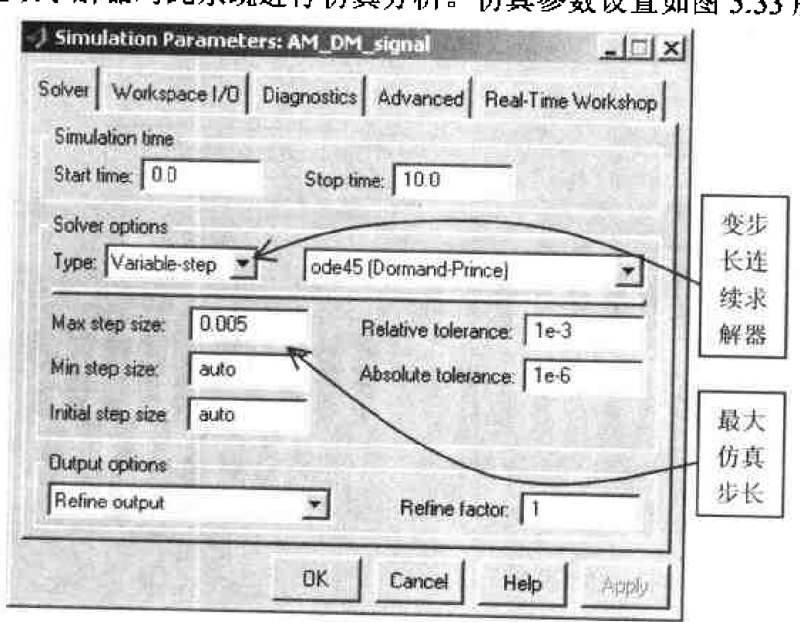


图 5.33　数字滤波器系统仿真参数设置

说明：使用变步长连续求解器是因为此系统为一个混合系统，其中既存在着离散部分（数字滤波器），又存在着连续部分（锯齿波信号的幅度调制），在混合系统仿真分析中将对此作进一步的解释。

下面对系统进行仿真，以分析数字滤波器的性能。将原始锯齿波信号与数字滤波器的输出信号（仿真结果）进行比较，如图5.34所示。

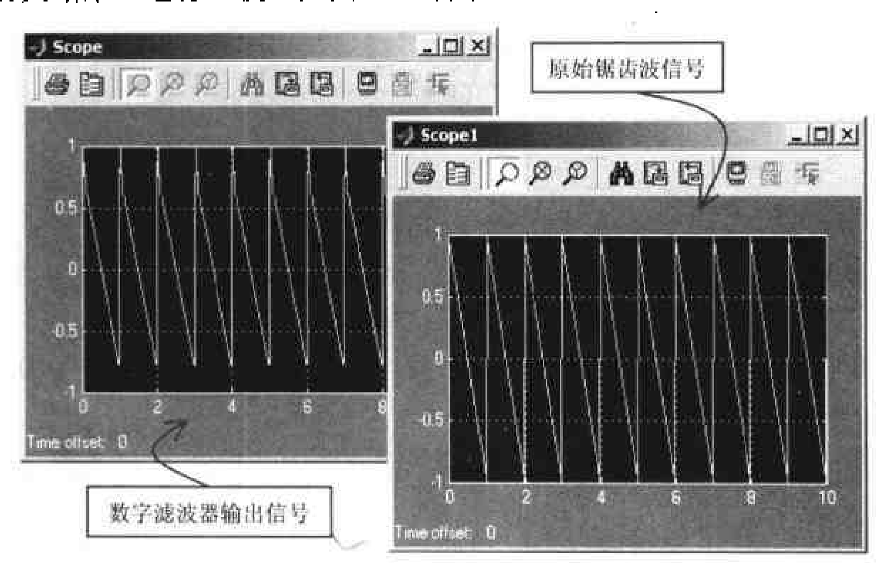


图5.34　系统仿真结果与原始锯齿波信号

很显然，数字滤波器的输出信号与原始锯齿波信号并不完全一致，而是存在着一定的失真。这种失真是不可避免的，因为实际并不存在理想滤波器，不能够完全滤除信号的某种频率的分量；而且在使用高频载波对低频信号进行调制时，信号之间不可避免地出现相互干扰。为了使读者对数字滤波器的功能有一个更为直观的认识，下面给出经高频正弦波调制后的信号与经过高频离散正弦波解调后的信号，如图5.35所示。从图中可以明显看出，此数字滤波器能够较好滤除解调后锯齿波信号的高频部分，从而获得低频锯齿波信号。

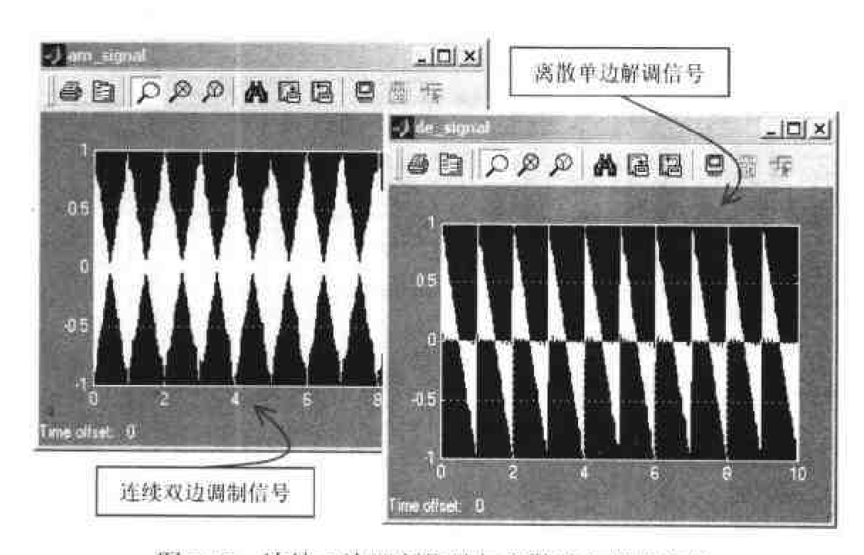


图5.35　连续双边调制信号与离散单边解调信号

5.5.2 线性连续系统仿真分析

5.4 节中对连续系统的仿真技术做了详细的介绍，对于具有连续状态的连续系统而言，往往使用积分器模块 Integrator 建立系统的模型。虽然实际的动态系统多数都是非线性系统，但是在一定的范围之内，非线性系统可以由线性系统来近似。由于非线性系统的设计、仿真与分析技术都比较复杂，而且还不成熟，尚无通用的理论形成；而线性系统的设计、仿真分析比较简单，故其应用非常广泛。本部分将介绍线性连续系统的仿真技术。

1. 建立线性连续系统的模型

建立线性连续系统模型的方法与建立一般线性系统模型的方法没有本质的区别。其不同之处在于，线性连续系统往往使用传递函数模型、零极点模型以及状态空间模型进行描述，因此在建立线性连续系统模型时，经常使用与此相应的模块（即 Continuous 模块库中的 Transfer Fcn 模块、Zero-Pole 模块以及 State-Space 模块）。下面以比例微分控制器系统模型的建立为例进行说明。

比例微分控制是一种早期的控制方法，它可以在系统出现位置误差之前，提前对系统产生修正作用，从而达到改善系统性能的目的。对比例微分控制的动态行为进行分析可知，比例微分控制系统是一种典型的二阶线性系统。下面分析比例微分控制系统在阶跃信号作用下的响应。

首先建立比例微分控制系统的模型，如图 5.36 所示。

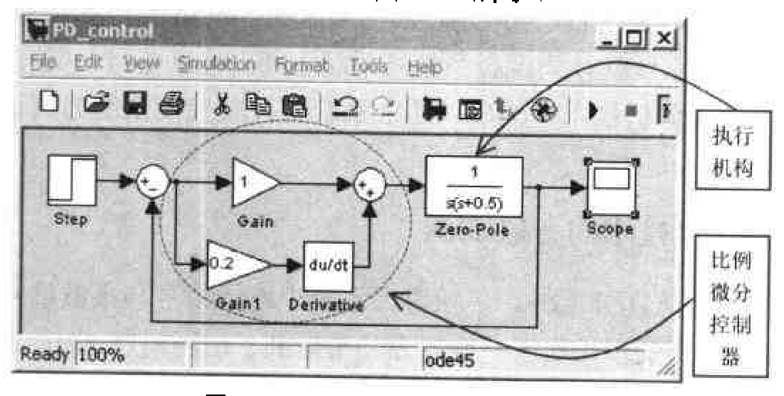


图 5.36　比例微分控制系统模型

2. 设置系统模块参数与仿真参数

在建立比例微分控制系统模型之后，需要设置各模块参数与系统仿真参数。系统模型中模块参数设置如下：

(1) Zero-Pole 模块设置：设置零点 Zeros 为 [ ]，设置极点 Poles 为[0　-0.5]，设置增益为 1。

(2) Step 信号模块设置：使用系统的默认取值，即单位阶跃信号。

(3) 其它各模块的参数设置如图 5.36 所示。

注意：如果使用积分器模块、状态空间模块或 LTI 模块建立线性系统模型，必须设置合适的

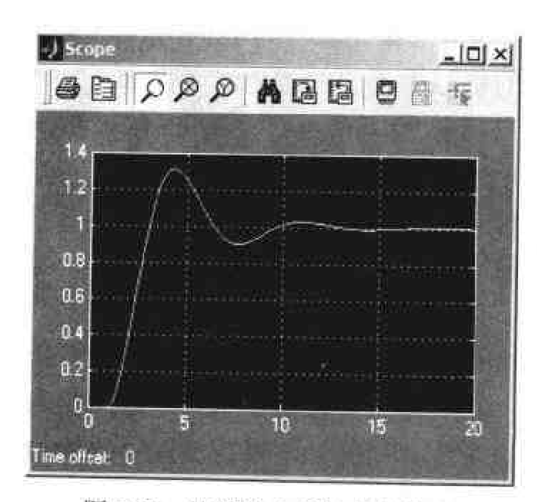


图 5.37　比例微分系统仿真结果

初始值，否则使用默认的初始值可能使系统进入不稳定的状态。

在设置系统模块参数之后，使用 Simulation Parameters 仿真参数对话框中的 Solver 选项卡设置系统仿真参数，如下所述：

(1) 仿真时间范围为 0~20 s。

(2) 使用变步长连续求解器（variable-step），仿真算法为 ode45。

(3) 最大仿真步长（Max step size）为 0.01。

(4) 绝对误差（Absolute tolerance）为 1e–6。

(5) 其余仿真参数使用默认取值。

3. 仿真分析

在对模块参数与系统仿真参数进行合适的设置之后，运行系统仿真，结果如图 5.37 所示。从系统的仿真结果可以明显看出，比例微分控制系统为典型的二阶线性系统：在阶跃信号的作用下，系统不断地对位置误差进行控制修正，最终使系统达到稳定的状态。为改善此系统的性能（如降低系统的超调量、减小系统的过渡时间等），可以对比例微分控制器进行调整（改变 Gain、Gain1 模块的增益）以获得更好的性能。

5.6 混合系统设计分析

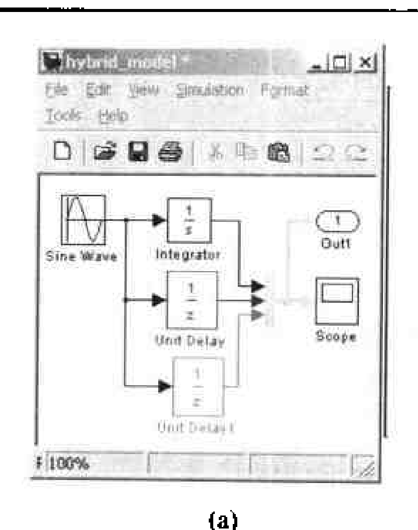
前面对一般的离散系统、连续系统以及线性系统的 Simulink 仿真技术做了详细的介绍。在实际的系统设计中，系统往往为混合系统（系统中既有连续信号，又有离散信号）。本节将介绍混合系统的仿真分析技术，然后以通信系统和行驶控制系统为例说明混合系统的设计、仿真与分析技术。

5.6.1 混合系统仿真技术的一般知识

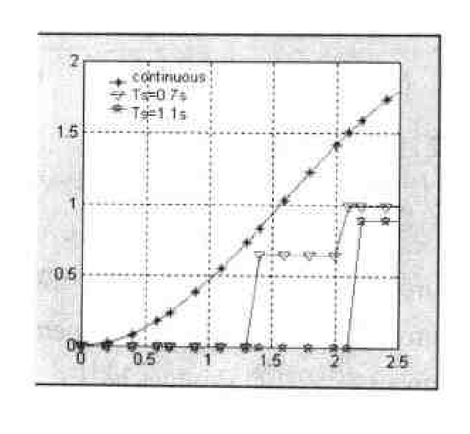
在对混合系统进行仿真分析时，必须考虑系统中连续信号与离散信号采样时间之间的匹配问题。Simulink 中的变步长连续求解器充分考虑到了上述问题。因此在对混合系统进行仿真分析时，应该使用变步长连续求解器。

由于混合系统中信号类型不一，使用同样的样式表示信号不利于用户对系统模型的理解。Simulink 仿真环境提供的 Sample time colors 功能可以很好地将不同类型、不同采样时间的信号用不同的颜色表示出来，从而可以使用户对混合系统中的信号有个清晰的了解。用户只需选择 Format 菜单中的 Sample time colors 命令便可实现这一功能。其中黑色的信号表示连续信号，其它颜色的信号表示离散信号；并且不同的颜色表示采样时间的不同，其中红色的信号表示其采样速率时间最快，绿色次之，而黄色表示含多速率的系统或信号。下面举例说明变步长连续求解器对信号采样时间的匹配问题，如图 5.38 所示。

图 5.38(a)所示为简单的混合系统模型（使用了 Sample time colors 功能表示不同采样时间的信号，其中 Unit Delay 模块采样时间为 0.7 s，Unit Delay1 模块采样时间为 1.1 s），图 5.38(b)所示为系统仿真输出数据点的图形，从中可以明显看出：变步长连续求解器不时地减小仿真步长以匹配离散信号的采样时刻。



(a)



(b)

图 5.38　变步长连续求解器对信号采样时间的匹配

5.6.2　混合系统设计之一：通信系统

前面以低通数字滤波器为例介绍线性离散系统仿真技术时，我们以无损信道通信系统为例说明了数字滤波器的功能。其实在实际的通信系统中，所有的信道都存在着不同程度的信道噪音，均使信道所传递的信号受到一定的损失。下面对一个实际的通信系统进行仿真分析，以说明混合系统的设计、仿真与分析。

1. 通信系统的物理模型与数学描述

1) 通信系统的信源

通信系统的信源为单位幅值与单位频率的低频锯齿波信号源，也就是通信系统所要传递的信号。

2) 通信系统的调制与解调

在此通信系统中，调制信号为正弦连续信号（幅值为 1，频率为 100 Hz），解调信号为正弦离散信号（幅值为 1，频率为 100 Hz，采样时间为 0.005 s），并且采用双边带抑制载波调制与解调（如 5.5 节中所述）。由于高频信号更容易受到噪音的干扰，而使信号出现较大的失真，故这里正弦信号的频率选择为 100 Hz，而非无损信道通信系统中所采用的 1000 Hz。

3) 通信信道

(1) 通信信道动态方程为$10^{-9}\ddot{y}+10^{-3}\dot{y}+y=u$。其中$u$为信道输入，$y$为信道输出。显然，此信道为一线性连续信道，信道传递函数描述如下：

$$\frac{Y(s)}{U(s)}=\frac{1}{10^{-9}s^2+10^{-3}s+1}$$

(2) 信道噪音：信道受到服从高斯正态分布的随机加性噪音的干扰，噪音均值为 0，方差为 0.01。

(3) 信道延迟：信道经过缓冲区为 1024 的延迟。

4) 数字滤波器

数字滤波器的差分方程为

$$y(n)-1.6y(n-1)+0.7y(n-2)=0.04u(n)+0.08u(n-1)+0.04u(n-2)$$

此数字滤波器为线性离散系统，使用滤波器形式对其进行描述如下：

$$\frac{Y(z)}{U(z)}=\frac{0.04+0.08z^{-1}+0.04z^{-2}}{1-1.6z^{-1}+0.7z^{-2}}$$

2. 建立通信系统模型

按照通信系统的物理与数学模型建立系统模型。在建立系统模型之前，首先给出建立系统模型所需要的系统模块，如下所述：

(1) Sources 模块库中的 Sine Wave 模块：作为高频载波信号与解调信号。

(2) Sources 模块库中的 Signal Generator 模块：产生低频锯齿波信号。

(3) Math 模块库中的 Product 模块：用于信号进行调制与解调。

(4) Continuous 模块库中的 Transfer Fcn 模块：描述通信信道。

(5) Sources 模块库中的 Random Number 模块：产生信道噪音。

(6) Continuous 模块库中的 Transport Delay 模块：产生信道延迟。

(7) Discrete 模块库中的 Discrete Filter 模块：描述数字滤波器。

(8) Subsystems 模块库中的 Subsystem 模块：封装系统中不同部分。

(9) Sinks 模块库的 Scope 模块：显示输出。

然后建立系统模型，并将信号幅值调制、通信信道、幅值解调封装到单独的子系统之中，如图 5.39 所示。

作者建议：在建立混合系统或复杂系统模型时，最好使用子系统技术将实现不同功能的模块组进行封装，并对各子系统进行简单的注释说明。这样非常有利于用户对系统模型的理解与维护。

3. 系统模块参数设置与仿真参数设置

在建立系统模型之后，按照系统的要求设置系统模块参数与仿真参数。用户对模块参数设置与仿真参数设置已经比较熟悉了，故在此仅给出各模块的参数与相应的仿真参数（所有没有给出的模块参数或仿真参数均使用系统的默认值）；用户可仿照前面的例子对参数进行正确的设置。

1) 信号调制子系统参数

(1) 正弦载波 Sine Wave 模块：频率 Frequency 为 100 Hz，幅值为 1。

(2) 锯齿信号 Signal Generator 模块：波形 Wave form 为 sawtooth（锯齿波）。

2) 通信信道子系统参数

(1) 随机信号 Random Number 模块：均值 Mean 为 0，方差 Variance 为 0.01。

(2) 信道延迟 Transfer Delay 模块：初始缓冲区 Initial buffer size 为 1024。

(3) 信道传递函数 Transfer Fcn 模块：分子 Numerator 为[1]，分母 Denominator 为[1e–9 1e–3 1]。

3) 信号解调子系统参数

正弦解调信号 Sine Wave1 模块：频率 Frequency 为 100 Hz，幅值为 1，采样时间 Sample time 为 0.005 s。

4) 数字滤波器参数

数字滤波器 Discrete Filter 模块参数：分子 Numerator 为[0.04 0.08 0.04]，分母 Denominator 为[1 –1.6 0.7]、采样时间 Sample time 为 0.005 s。

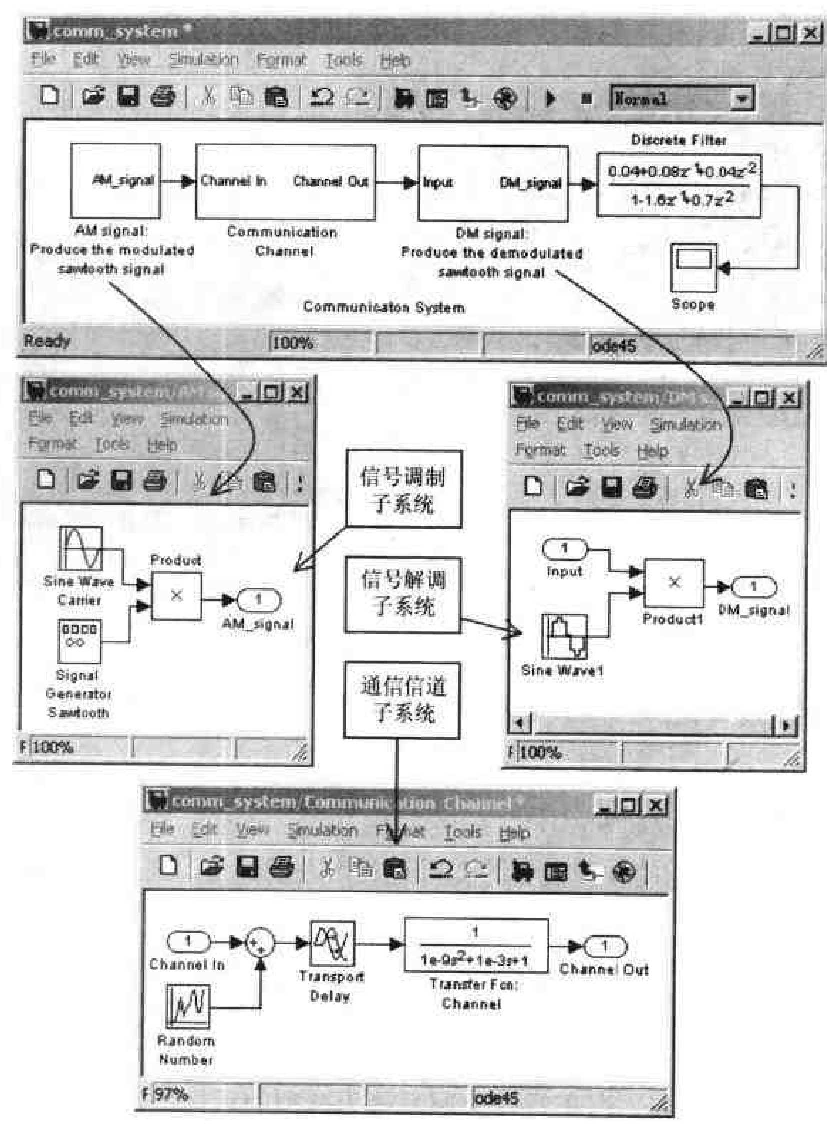


图 5.39　通信系统模型

5) 系统仿真参数

(1) 系统仿真时间：从 0 至 10 s。

(2) 仿真求解器：变步长连续求解器。

(3) 绝对误差：1e–6。

(4) 最大仿真步长：0.01。

4. 系统仿真与分析

在对系统模块参数与系统仿真参数设置之后，接下来对系统进行仿真分析。为了对通信系统的整体性能有一个直观的认识，这里将系统仿真结果（即通信系统的输出信号）与原始的锯齿波信号（即通信系统所要传递的信号）进行比较，如图 5.40 所示。从图中可以看出，由于通信信道的延迟以及加性随机噪音的干扰，使得通信系统的输出信号比原始锯齿波信号的起始时间慢 1 s，而且存在一定的失真，但只要失真小于一定的阈值，便不会对锯齿波信号的使用造成太大的影响。

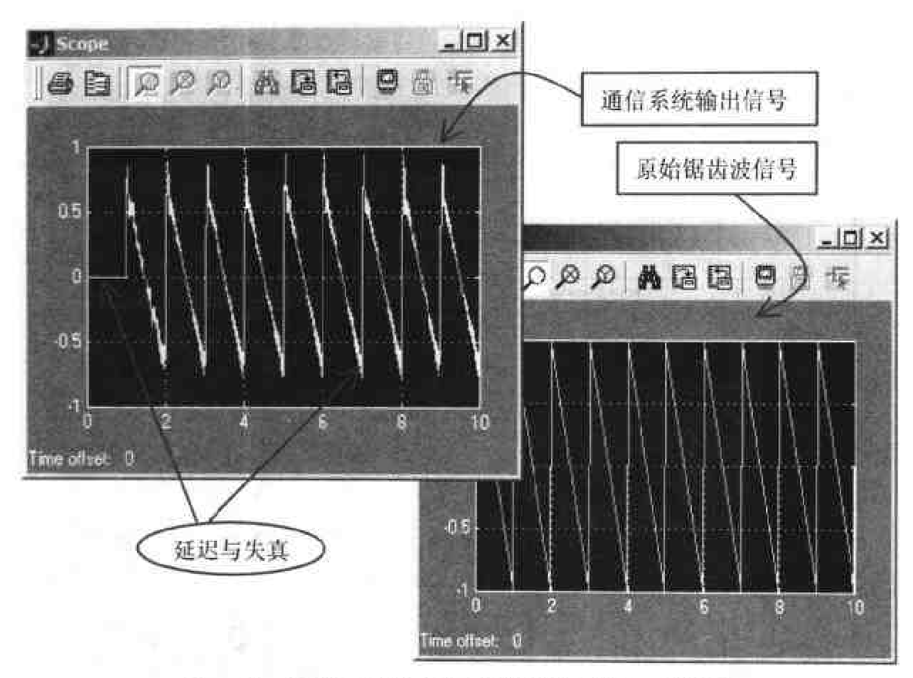


图 5.40 通信系统输出与原始锯齿波信号的比较

5.6.3 混合系统设计之二：行驶控制系统

汽车行驶控制系统是应用非常广泛的控制系统之一，其主要目的是对汽车速度进行合理的控制。系统的工作原理如下：

(1) 汽车速度操纵机构的位置发生改变以设置汽车的速度，这是因为操纵机构的不同位置对应着不同的速度。

(2) 测量汽车的当前速度，并求取它与指定速度的差值。

(3) 由速度差值信号驱动汽车产生相应的牵引力，并由此牵引力改变汽车的速度直到其速度稳定在指定的速度为止。

由系统的工作原理来看，汽车行驶控制系统为典型的反馈控制系统。下面建立此系统的 Simulink 模型并进行仿真分析。

1. 汽车行驶控制系统的物理模型与数学描述

1) 速度操纵机构的位置变换器

位置变换器是汽车行驶控制系统的输入部分，其目的是将速度操纵机构的位置转换为相应的速度，二者之间的数学关系如下所示：

$$v = 50x + 45, \quad x \in [0, 1]$$

其中 x 为速度操纵机构的位置，v 为与之相应的速度。

2) 离散行驶控制器

行驶控制器是整个汽车行驶控制系统的核心部分。简单来说，其功能是根据汽车当前速度与指定速度的差值，产生相应的牵引力。行驶控制器为一典型的 PID 控制器，其数学描述为：

积分环节：$x(n) = x(n-1) + u(n)$

微分环节：$d(n) = u(n) - u(n-1)$

系统输出：$y(n) = Pu(n) + Ix(n) + Dd(n)$

其中 $u(n)$ 为系统输入，相当于汽车当前速度与指定速度的差值。$y(n)$ 为系统输出，相当于汽车牵引力，$x(n)$ 为系统中的状态。P、I 与 D 为 PID 控制器的比例、积分与微分控制参数，其取值分别为 $P=1$，$I=0.01$，$D=0$。

3) 汽车动力机构

汽车动力机构是行驶控制系统的执行机构。其功能是在牵引力的作用下改变汽车速度，使其达到指定的速度。牵引力与速度之间的关系为

$$F = m\dot{v} + bv$$

其中 v 为汽车的速度，F 为汽车的牵引力，m=1000 kg 为汽车的质量，$b=20$ 为阻力因子。

2. 建立汽车行驶控制系统的模型

按照汽车行驶控制系统的物理模型与数学描述建立系统模型。在进行系统模型之前，首先给出建立系统模型所需的主要系统模块：

(1) Math 模块库中的 Slider Gain 滑动增益模块：对位置变换器的输入信号 x 的范围进行限制。

(2) Discrete 模块库中的 Unit Delay 单位延迟模块：用来实现行驶控制器（即 PID 控制器）。

(3) Continuous 模块库中的 Integrator 积分器模块：用来实现汽车动力机构。

(4) Subsystems 模块库中的 Subsystem 子系统模块：用来对系统不同的部分进行封装。

然后建立系统模型，并将速度操纵机构的位置变换器、行驶控制器、汽车动力机构封装到不同的子系统之中，如图 5.41 所示。

3. 系统模块参数设置与仿真参数设置

在建立系统模型之后，按照系统的要求设置系统模块参数与仿真参数。这里仅给出模块参数与相应的仿真参数。

1) 速度操纵机构的位置变换器参数

(1) Slider Gain 模块：最小值 Low 为 0，最大值 High 为 1，初始取值 0.555。

(2) Gain 模块：增益取值为 50。

(3) Constant1 模块：常数取值为 45。

2) 行驶控制器参数

(1) 所有 Unit Delay 模块：初始状态为 0，采样时间为 0.02 s。

(2) P、I、D 增益模块：取值分别为 1、0.01、0。

3) 汽车动力机构参数

(1) Gain 模块：取值为 1/m，即 1/1000。

(2) Gain1 模块：取值为 b/m，即 20/1000。

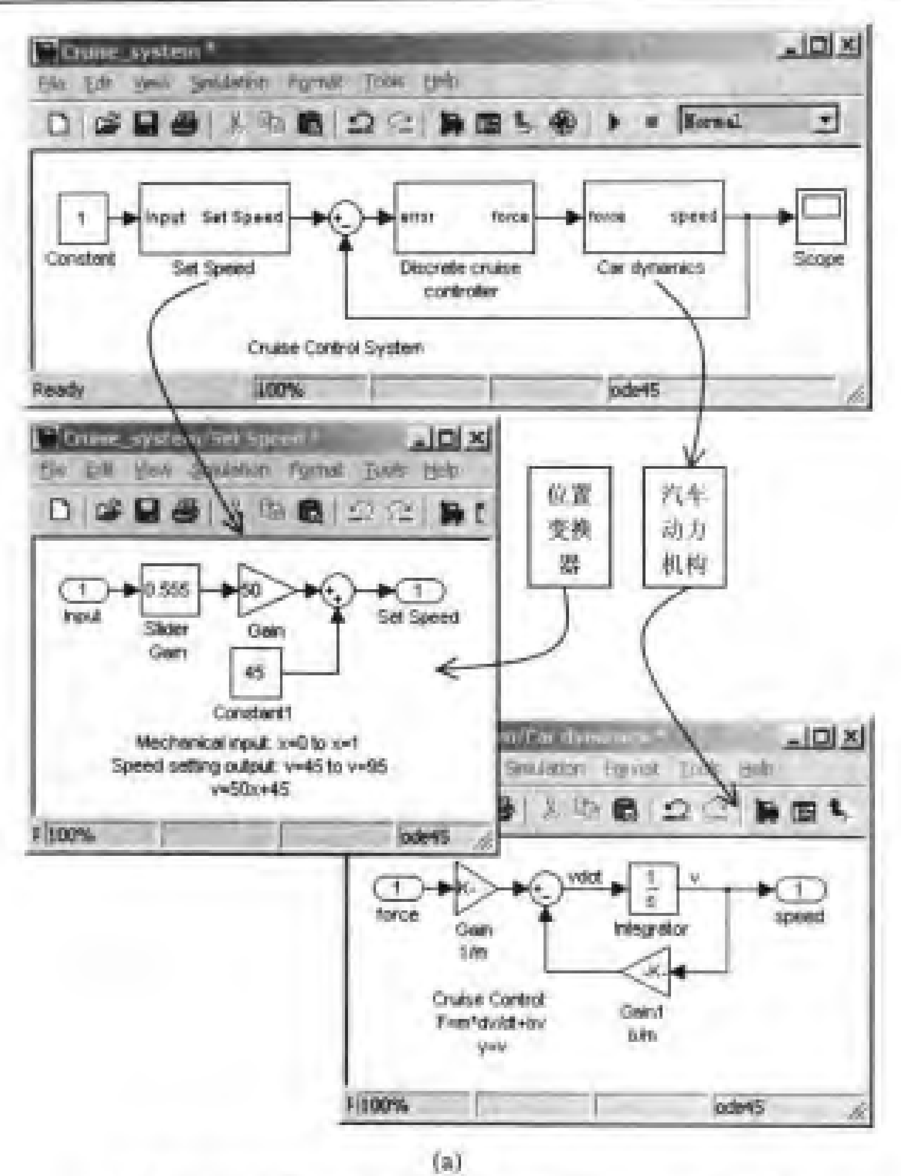
(3) Integrator 模块：初始状态为 0，即速度初始值为 0。

4) 系统仿真参数

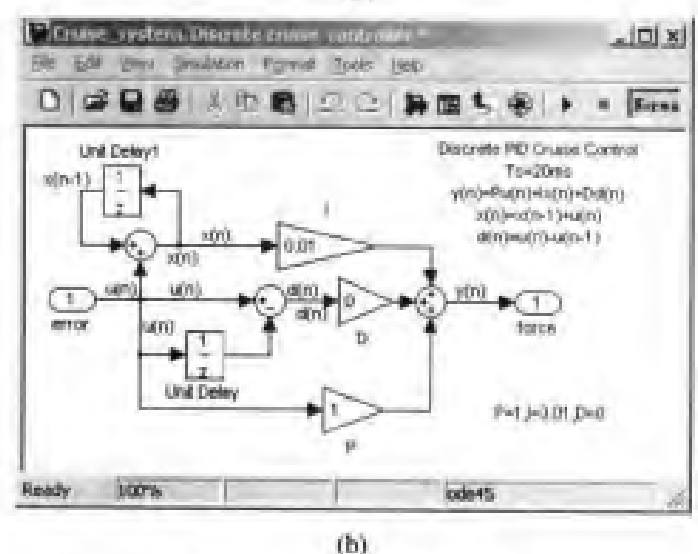
(1) 仿真时间范围：从 0 至 1000 s。

(2) 求解器：使用变步长连续求解器。

注意：其余模块参数与仿真参数均使用默认取值。



(a)



(b)

图 5.41 汽车行驶控制系统

(a) 位置变换器与汽车动力机构； (b) 行驶控制器

4. 系统仿真与分析

在对系统模块参数与系统仿真参数设置之后，对系统进行仿真分析。为了使用户对离散行驶控制器的作用有一个直观的认识，这里使用两组不同的 PID 控制参数对系统进行仿真，其结果如图 5.42 所示。

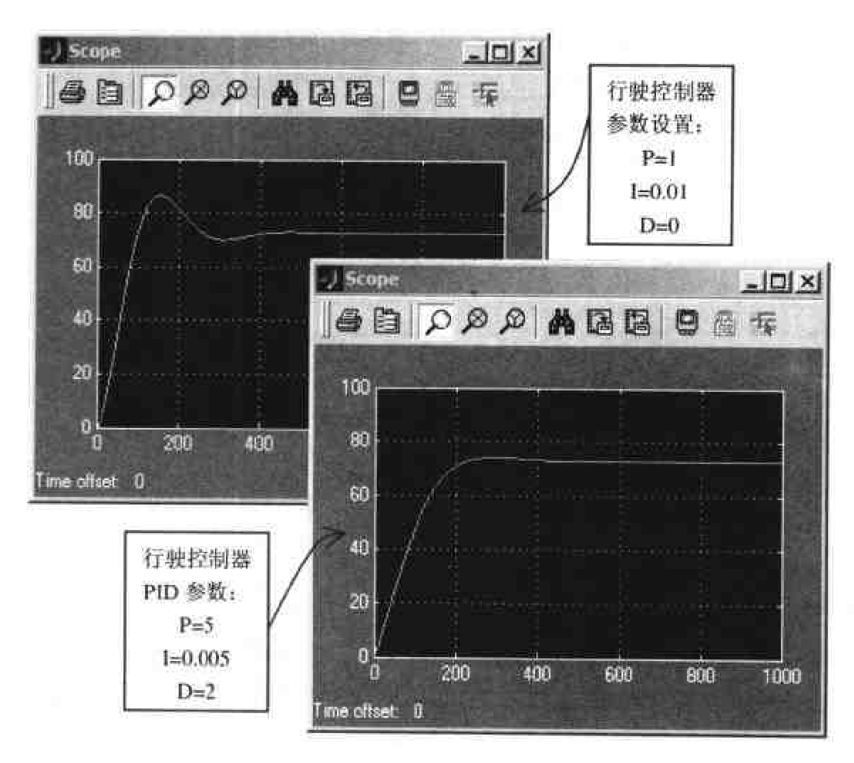


图 5.42　不同控制参数下的仿真结果

汽车行驶控制系统的目的是使汽车的速度在较短的时间内平稳地达到指定的速度。从图 5.42 的仿真结果中可以看出，在行驶控制器控制参数取值为 P=1、I=0.01、D=0 时，汽车的速度并非直接达到指定的速度，而是经过一个振荡衰减过程，最后逐渐过渡到指定的速度。此时行驶控制系统为典型的二阶欠阻尼控制系统。

对于 PID 控制器而言，增加微分控制参数 D 可以减小系统超调量，缩短系统调节时间；增加积分控制参数 I 可以增加系统超调量，延长系统调节时间；而增加比例控制参数 P 值可以缩短系统调节时间。由于行驶控制器为一离散 PID 控制器，所以适当增加控制器参数的 P、D 取值，减小 I 取值可以改善系统的性能，这一点可以从行驶控制系统在控制参数 P=5、I=0.005、D=2 下的仿真结果中明显看出。

5.6.4　工作空间输入输出 Workspace I/O 设置

在对动态系统进行仿真分析时，往往需要对系统的仿真结果进行进一步的定量分析。使用 Scope 模块可以直接显示系统仿真结果，非常有利于对系统的定性分析，但并不利于系统的定量分析。本书第 4 章中介绍的 Simulink 与 MATLAB 的接口设计技术，允许 Simulink 与 MATLAB 之间进行数据交互，如从 MATLAB 工作空间中获得系统模块参数、输出仿真结果至 MATLAB 工作空间等。

除了使用上面的方法进行 Simulink 与 MATLAB 的数据交互之外，用户还可以使用仿真参数对话框中的 Workspace I/O 选项卡对动态系统仿真数据的处理进行更高级的控制。图 5.43 所示为仿真参数设置中的 Workspace I/O 选项卡。

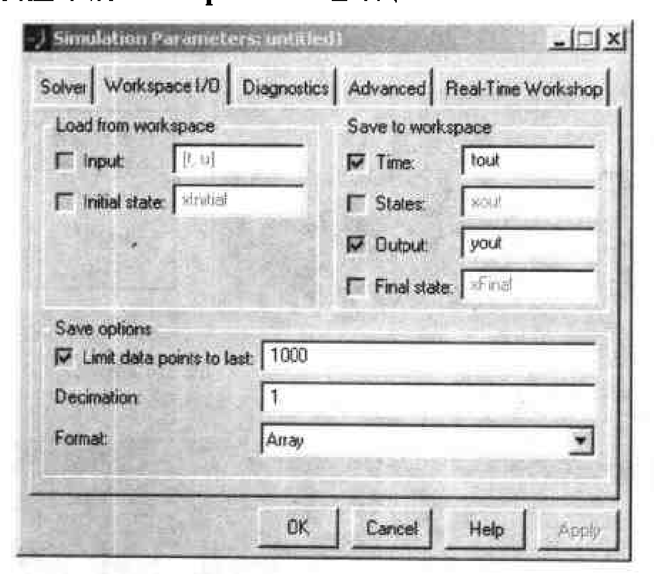


图 5.43 Workspace I/O 选项卡

下面对 Workspace I/O 选项卡的功能与使用做一个简单的介绍。

1. 从 MATLAB 工作空间获得系统输入（Load form workspace）

虽然 Simulink 提供了多种系统输入信号，但并不能完全满足需要。Simulink 允许使用用户自定义的信号作为系统输入信号。在 Load form workspace 框中，用户可以设置 MATLAB 中的变量作为系统输入信号或系统状态初始值，如下所述：

(1) Input：用来设置系统输入信号。其格式为[t，u]，其中 t、u 均为列向量，t 为输入信号的时间向量，u 为相应时刻的信号取值，可以使用多个信号输入，如[t，u1，u2]。输入信号与 Simulink 的接口由 Inport 模块（In1 模块）实现。

(2) Initial state：用来设置系统状态变量初始值。初始值 xInitial 可为行向量。

注意： *使用 xInitial state 所设置状态变量初始值会自动覆盖系统模块中的设置。另外，输入信号与状态变量需要按照系统模型中 Inport 模块（即 In1 模块）的顺序进行正确设置。*

2. 输入数据至 MATLAB 工作空间（Save to workspace）

使用 Workspace I/O 选项卡可以将系统的仿真结果、系统仿真时刻、系统中的状态或指定的信号输出到 MATLAB 工作空间中，以便用户对其进行定量分析，如下所述：

(1) Time：输出系统仿真时刻。

(2) States：输出系统模型中所有的状态变量。

(3) Output：输出系统模型中所有由 Outport 模块（即 Out1 模块）表示的信号。

(4) Final state：输出系统模型中的最终状态变量取值，即最后仿真时刻处的状态值。

3. 数据保存设置（Save option）

(1) Limit data points to last：表示输出数据的长度（从信号的最后数据点记起）。

(2) Format：表示输出数据类型，共有三种形式：Structure with Time（带有仿真时间变量的结构体）、Structure（不带仿真时间变量的结构体）以及 Array（信号数组）。

上面简单介绍了 Workspace I/O 选项卡的设置与功能，下面举例说明。对于图 5.44 所示的系统，按照图中所示设置系统仿真参数中的 Workspace I/O 选项卡，以从 MATLAB 工作空间获取系统输入信号、系统状态初始值并将系统仿真结果输出至 MATLAB 工作空间之中。

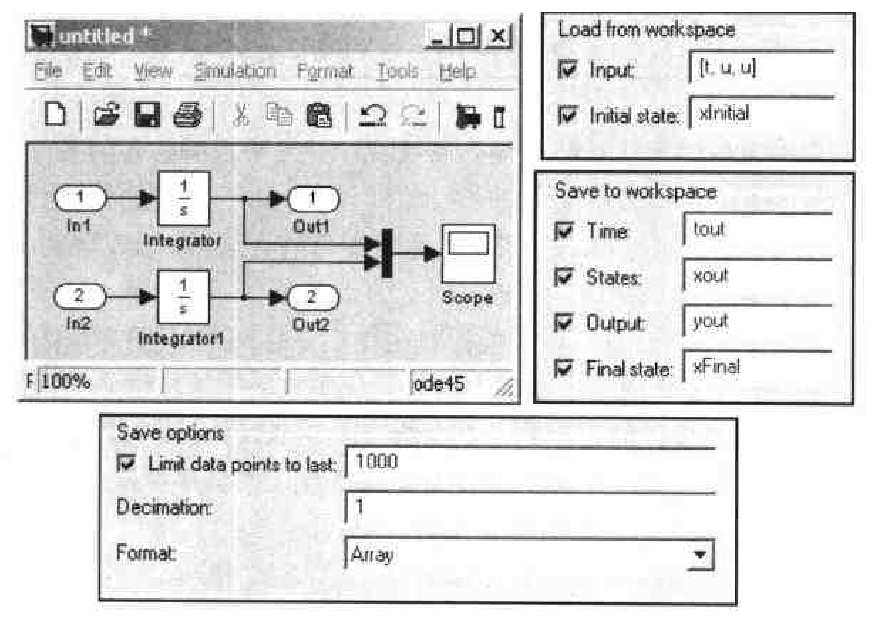


图 5.44 Workspace I/O 设置举例

在运行仿真之前，首先需要生成系统输入信号与状态初始值，在 MATLAB 命令窗口中键入如下命令：

```
>> t=1:0.1:10; t=t';
>> u=sin(t);   u=u';
>> xInitial=[0,1];
```

然后运行系统仿真，为了观察 Workspace I/O 设置的效果，这里使用 Scope 模块显示仿真结果，如图 5.45 所示。

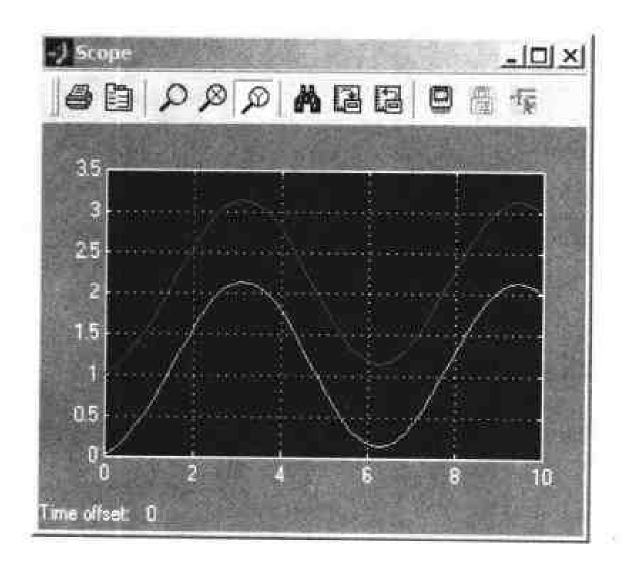


图 5.45 使用 Workspace I/O 设置的仿真结果

此时查看 MATLAB 的工作空间，用户可以发现系统仿真结果（仿真时间、系统状态、系统运算结果以及最终状态等）被正确输出到 MATLAB 工作空间中，这里不再赘述。

5.7 Simulink 的调试技术

功能强大、界面友好的调试功能是优秀系统设计开发平台所必备的条件之一。Simulink 作为高性能的系统设计、仿真与分析平台，给用户提供了强大的模型调试工具。通过 Simulink 的调试工具，用户可以对动态系统的系统模型进行调试，以发现其中可能存在的问题，然后进行修改，从而快速完成系统设计、仿真与分析的目的。

不同领域中的不同系统模型，其复杂程度往往相差悬殊，对系统模型调试的复杂程度也大不相同。Simulink 所提供的图形调试器可以满足多数应用领域中系统模型的调试，而并非针对专门的应用领域所设计的。因此，在介绍 Simulink 调试器功能时，这里仅以最简单的例子对其进行说明，以使具有不同专业背景的系统设计人员都可以很好理解。

5.7.1 Simulink 图形调试器启动

Simulink 的图形调试器具有优秀的用户界面。使用菜单 Tools 下的 Simulink debugger 命令或使用调试器按钮可以启动调试器。图 5.46 所示为 Simulink 图形调试器窗口。

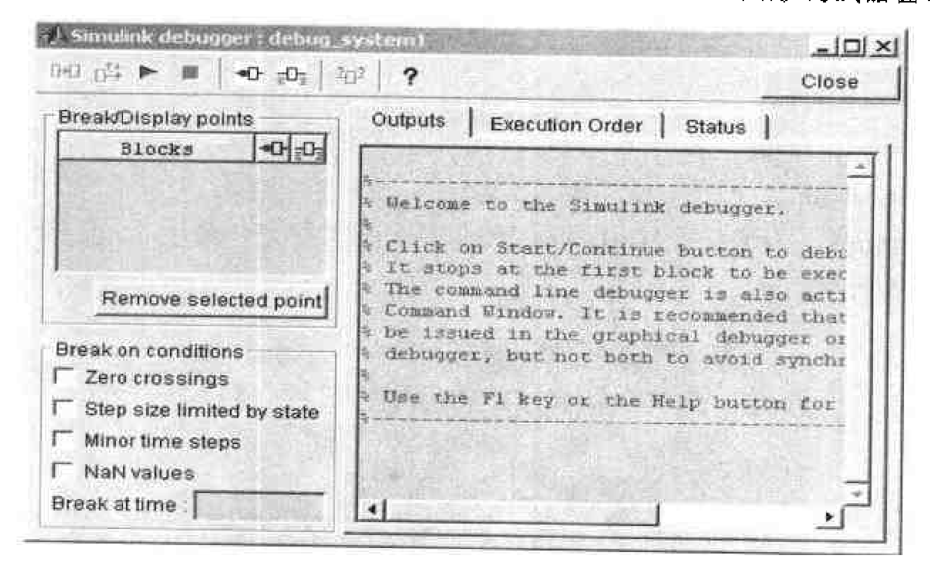


图 5.46 Simulink 调试器窗口

5.7.2 调试器的操作设置与功能

启动 Simulink 调试器，设置合适的调试断点之后，便可以对系统模型中指定的模块或信号进行调试。在设置断点进行调试之前，首先对 Simulink 图形调试器中的操作设置与功能做一个简单的介绍。

1. Simulink 调试器工具栏

Simulink 调试器工具栏命令功能介绍如图 5.47 所示。

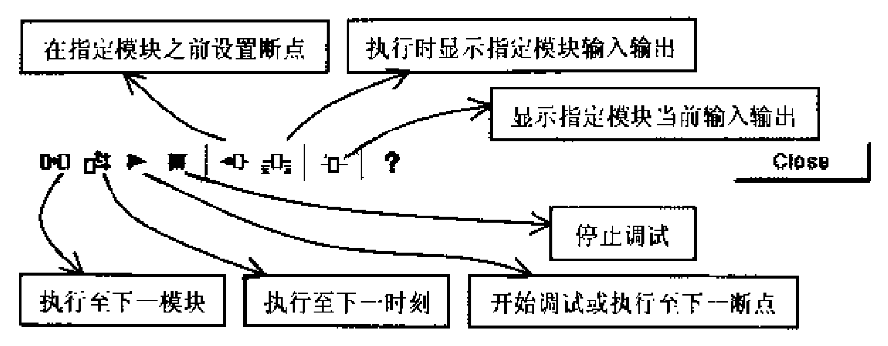


图 5.47　Simulink 调试器工具栏命令介绍

调试器工具栏命令的功能都比较简单，但有一点需要说明，执行至下一模块与执行至下一仿真时刻是有区别的。执行至下一模块所需的时间可以大于也可以小于 Simulink 相邻仿真时刻之间的差值，二者一般并不相等。

2. 断点显示及断点条件设置

Simulink 提供了友好的调试界面，用户可以在断点显示框中了解到当前断点的信息，如断点位置、断点模块的输入输出等，如图 5.48 所示。

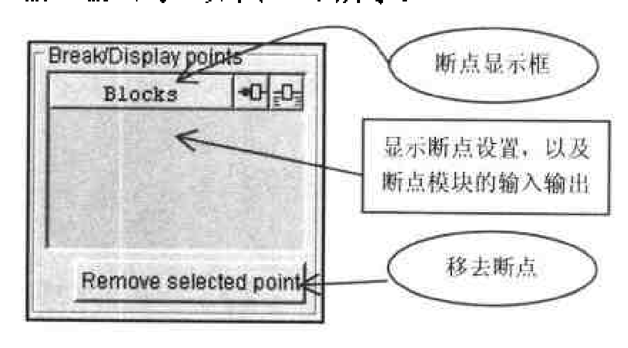


图 5.48　断点显示框

一般来说，用户可以于调试前在指定的模块之前设置断点。但是多数情况下，用户需要在一定的条件下设置系统断点以进行调试。Simulink 调试器提供了五种断点条件设置，如图 5.49 所示。

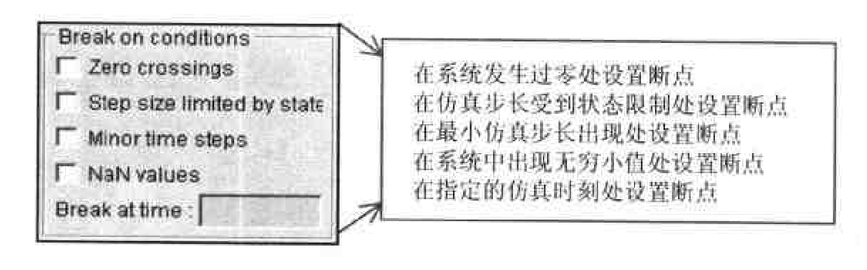


图 5.49　断点条件设置

3. 调试器输出窗口

在对指定的系统模型进行调试时，调试结果均在 Simulink 的输出窗口显示。图 5.50 所示为 Simulink 的调试器输出窗口。

下面对其进行简单的介绍：

(1) Outputs：输出调试结果，如调试时刻、调试的模块以及模块输入输出等。

(2) Execution Order：输出调试顺序，即调试过程中各模块的执行顺序。

(3) Status：输出调试状态，如当前仿真时间、缺省调试命令（执行至下一模块或执行至下一时间步）、调试断点设置以及断点数等状态信息。

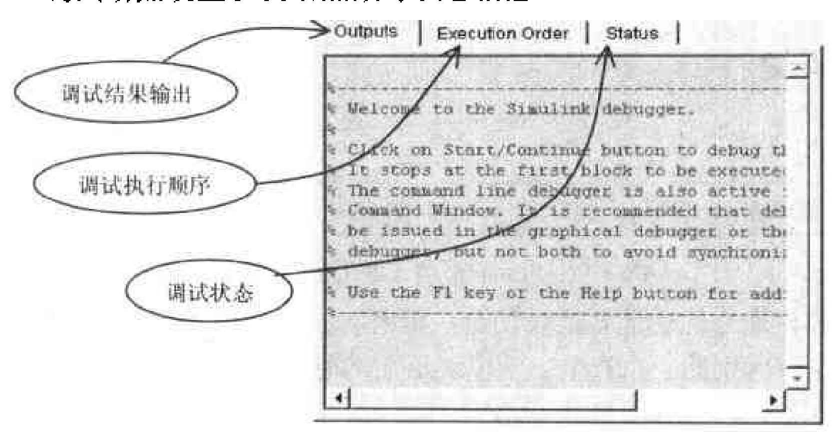
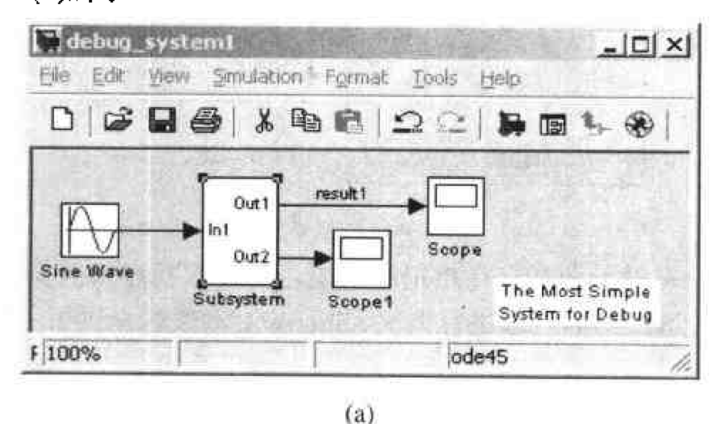


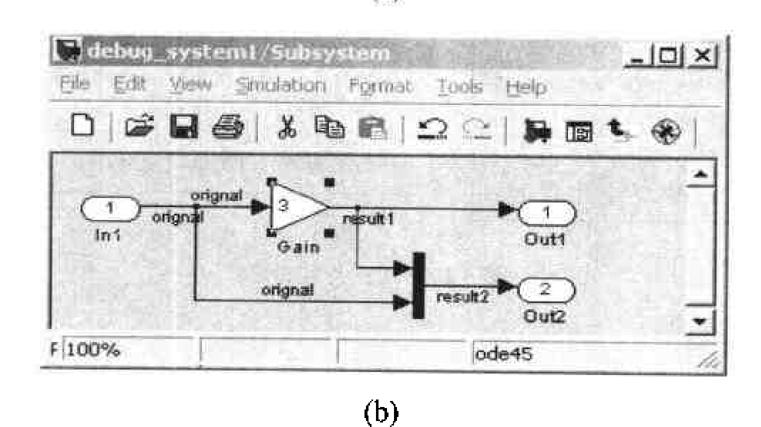
图 5.50 调试器输出窗口

5.7.3 系统调试举例

从对调试器操作设置与功能的介绍中可以看出，使用 Simulink 图形调试器既简单又直观。下面以图 5.51 所示的系统模型为例说明 Simulink 的调试技术，系统模型如图 5.51(a)所示，子系统如图 5.51(b)所示。



(a)



(b)

图 5.51 系统模型及其子系统

此系统的结构与功能都非常简单，虽然使用 Simulink 的调试器对其进行调试有些"大材小用"，但对介绍 Simulink 的调试功能来说，它已经足够了。

1. 设置系统调试断点

(1) 启动 Simulink 的调试器。

(2) 在子系统中增益模块（Gain 模块）之前设置断点：选择 Gain 模块，然后单击 Simulink 调试器工具栏中的"在指定模块之前设置断点"图标即可。此时，断点显示框中将显示断点设置情况，如图 5.52 所示。

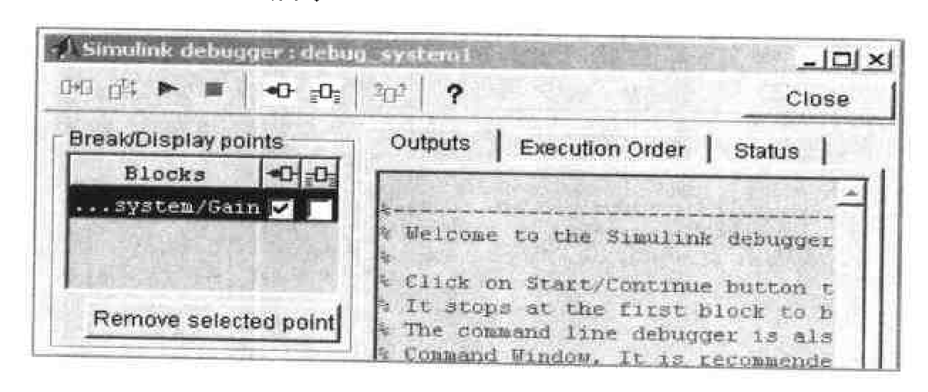


图 5.52　系统调试断点设置

当然，用户也可以按照一定的断点条件来设置系统调试断点，这一点也是非常容易的，读者可以尝试一下。

2. 逐模块执行调试

系统模型一旦进入调试模式，使用调试器工具栏中的第一个按钮"执行下一模块"，便可逐模块对系统进行调试。在调试过程中，即将被执行的模块系统会用黄颜色突出显示。当此模块被执行完毕时，在调试器输出窗口中将显示相应的系统仿真时刻、模块的输入与输出。如果系统调试至系统模型中的子系统，则调试器会自动打开相应的子系统。图 5.53 所示为逐模块执行系统调试的示意图。

注意：*在系统调试时，Inport 模块（In1 模块）、Outport 模块（Out1 模块）以及 Mux 模块等"虚模块"均被调试器忽略。这是因为虚模块只是用来表示信号的某种操作，并不真正执行。*

下面对图 5.53 所示示意图做个简单的说明。当系统进入逐步调试模式时，模块 Sine Wave 首先被执行（以黄色高亮显示），此时在调试器输出窗口中显示如下信息：系统仿真时刻（其取值 Tm 为 0），即将被执行的模块（Sine Wave 模块）；然后单击调试器工具栏"执行至下一模块"图标，调试器将自动打开子系统并将执行模块 Gain（以黄色高亮显示），此时调试器输出窗口中将显示如下信息：模块 Sine Wave 的输出（其值为 0）、即将执行的模块（Gain 模块）。如此反复进行逐模块系统调试。

3. 逐时间步执行调试

逐模块调试系统可使用户对系统中任何模块的输入输出进行详细的观测，但是此调试方式非常耗时，尤其是对大型复杂系统进行调试。Simulink 允许用户使用逐时间步方式对系统进行调试。逐时间步调试的特点是调试的时间间隔与系统仿真步长一致，并且在系统断点处停止执行。图 5.54 为逐时间步执行系统调试的示意图。

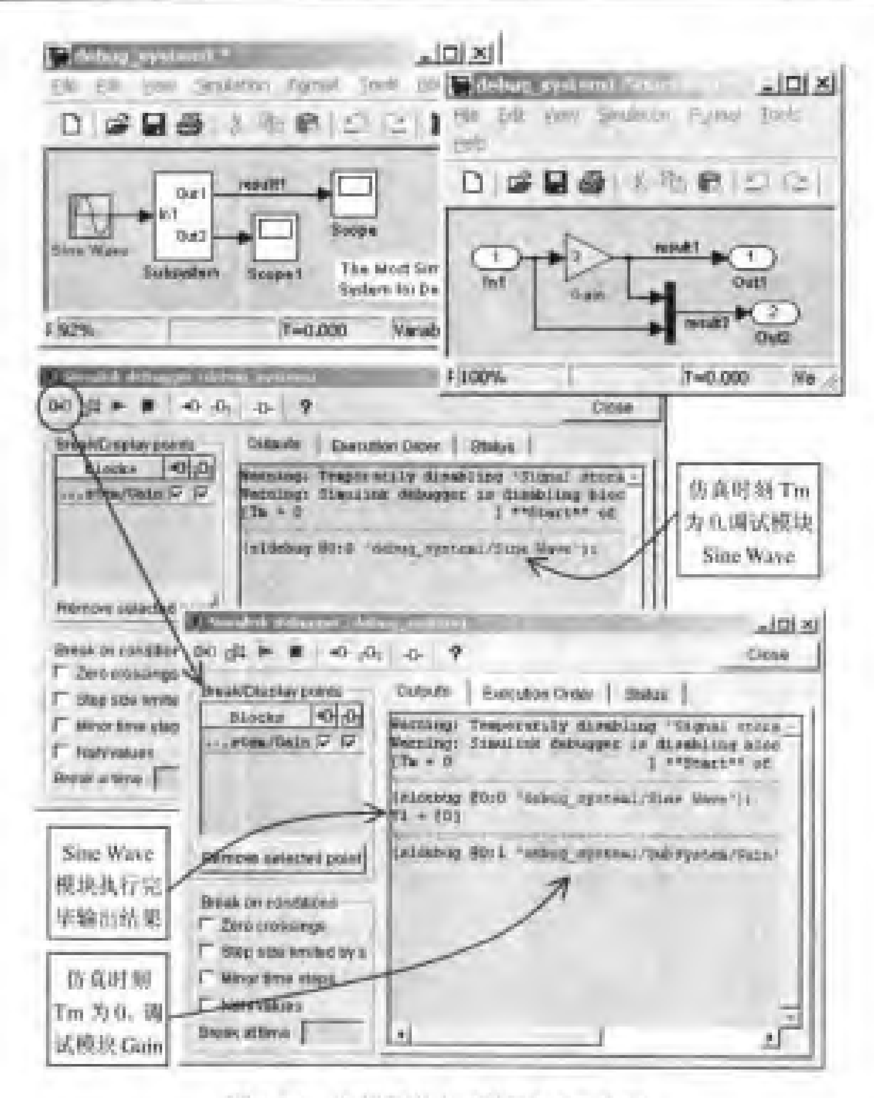


图 5.53　逐模块执行系统调试示意图

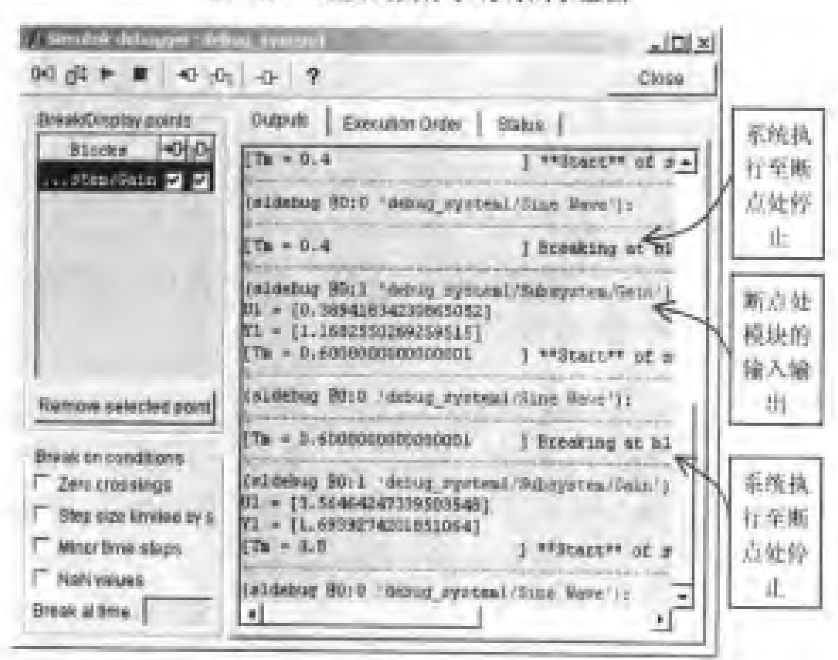


图 5.54　逐时间步执行系统调试示意图

下面对图 5.54 所示的示意图进行简单的说明。在逐时间步调试模式下，系统调试时间间隔与仿真步长一致，这里给出系统在仿真时刻 Tm=0.4 s、0.6 s 与 0.8 s 之间的调试结果。当系统调试执行至仿真时刻 0.4 s 之后，到断点模块（Gain 模块）处停止执行；然后单击“逐时间步调试”图标继续进行调试，在调试输出窗口中显示断点模块在上一仿真时刻（0.4 s）的输入输出结果，并在断点模块处停止执行（此时仿真时刻为 0.6 s）。如此反复进行系统逐时间步调试。

4．调试执行顺序与调试状态

在系统调试过程中，Simulink 调试器按照一定的顺序对系统模块进行调试。使用调试器输出窗口中的 Execution Order 窗口可以显示系统模块的调试顺序。此功能对于简单的系统调试而言作用不大，但对于大型多速率复杂系统来说是非常重要的。另外，如果系统中包含代数环，在 Execution Order 窗口中也会显示相关的系统模块。

此外，用户还可以在调试过程中随时对调试器所处的状态进行观察，此时需要使用调试器输出窗口中的 Status 窗口。图 5.55 所示为上述系统调试中模块的执行顺序与调试时的状态。

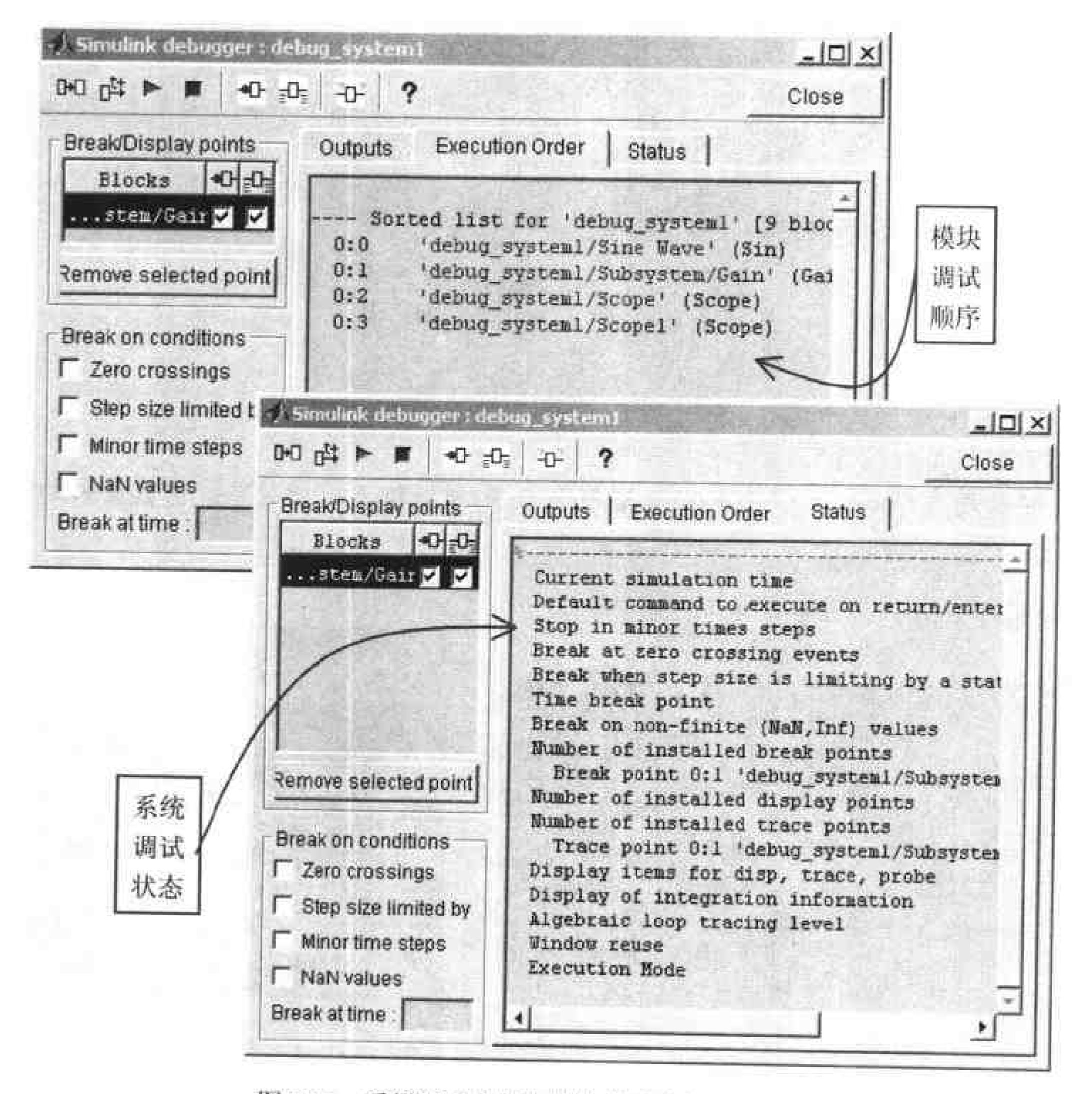
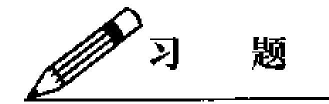


图 5.55　系统调试中模块的执行顺序与调试状态



习　题

图 5.56 所示为弹簧-质量-阻尼器机械位移系统。请建立此动态系统的 Simulink 仿真模型，然后分析系统在外力 $F(t)$ 作用下的系统响应(即质量块的位移 $x(t)$)。其中质量块质量 m=5 kg，阻尼器的阻尼系数 $f=0.5$，弹簧的弹性系数 $K=5$；并且质量块的初始位移与初始速度均为 0。

说明：外力 $F(t)$ 由用户自己定义，目的是使用户对系统在不同作用下的性能有更多的了解。

提示：(1) 首先根据牛顿运动定律建立系统的动态方程，如下式所示：

$$m\frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2}+f\frac{\mathrm{d}x(t)}{\mathrm{d}t}+kx(t)=F(t)$$

(2) 由于质量块的位移 $x(t)$ 未知，故在建立系统模型时，使用积分模块 Integrator 对位移的微分进行积分以获得位移 $x(t)$，且积分器初始值均为 0。

图 5.56　弹簧-质量-阻尼器机械位移系统示意图

为建立系统模型，将系统动态方程转化为如下的形式：

$$\frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2}=\frac{F(t)}{m}-\frac{k}{m}x(t)-\frac{f}{m}\frac{\mathrm{d}x(t)}{\mathrm{d}t}$$

然后以此式为核心建立系统模型。

参考答案：

(1) 系统模型如图 5.57 所示。

(2) 系统仿真结果。设外力 $F(t)$ 为幅值为 5 的阶跃输入，此时系统仿真结果如图 5.58 所示。

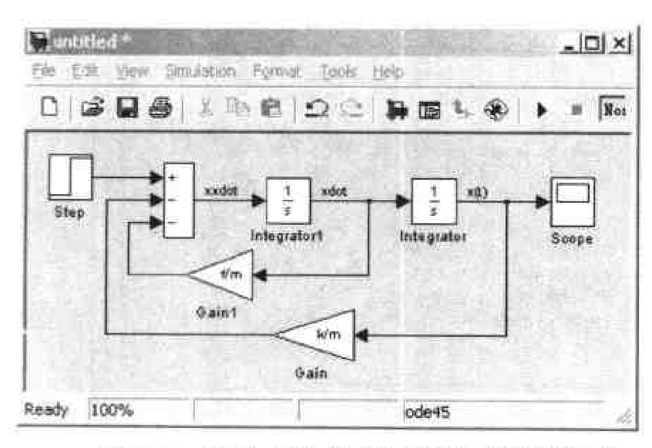


图 5.57　弹簧-质量-阻尼器机械位移系统模型

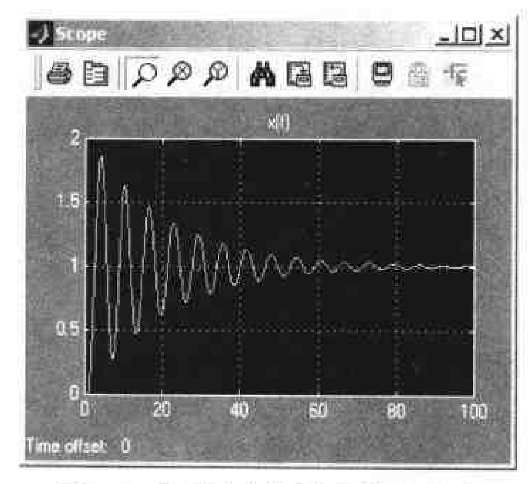


图 5.58　阶跃外力作用下系统仿真结果

第三部分

精通

Simulink 高级仿真技术

虽然在第二部分中对动态系统建模、仿真与分析做了详细的介绍，但这对于大型复杂动态系统的建模、仿真与分析来说还略显不足。第三部分将主要介绍 Simulink 的高级仿真技术。读者通过对木部分的学习，可以得心应手地对复杂动态系统建模、仿真与分析，并可对 Simulink 的使用进行高级控制，同时也可根据需要扩展 Simulink 的功能。本部分包括如下内容:

➤Simulink 系统仿真原理：主要介绍 Simulink 的仿真求解器、过零事件、代数环、高级积分器以及高级仿真参数设置。

➤Simulink 子系统技术：主要介绍 Simulink 的高级子系统技术、子系统封装技术以及 Simulink 模块库技术。

➤Simulink 命令行仿真技术：主要介绍使用命令行方式进行动态系统仿真分析。

➤S-函数：主要介绍 Simulink 系统函数 System Function（S-函数）的基本概念、工作原理与函数编写技术。

第 6 章　Simulink 系统仿真原理

内容概要

· Simulink 求解器概念
· 过零事件的产生与解决方案
· 代数环的产生与解决方案
· 高级积分器的概念
· 仿真参数：高级选项与诊断选项

本书在第二部分对使用 Simulink 进行动态系统模型建立、系统仿真及分析进行了详细的介绍。对于一般的用户，使用这些知识便能够对大部分的动态系统进行建模、仿真与分析。但是对于高级系统设计人员来说，熟悉 Simulink 对动态系统进行仿真的工作原理必定会对系统的设计、仿真与分析起到很好的作用。前面所介绍的仅仅是 Simulink 仿真平台的使用方法，用户完全可以在较短的时间内熟练掌握。本章将对 Simulink 系统仿真原理作简单的介绍，以使用户对 Simulink 进行系统仿真的核心有一个简单的了解。这对系统分析与设计的作用不言而喻。

6.1　Simulink 求解器概念

Simulink 求解器是 Simulink 进行动态系统仿真的核心所在，因此欲掌握 Simulink 系统仿真原理，必须对 Simulink 的求解器有所了解。在第 5 章中讲述动态系统的 Simulink 仿真技术时曾简单提及 Simulink 求解器的选择与使用，本节将对其作深入的介绍。

6.1.1　离散求解器

第 3 章中简单介绍了动态系统的模型及其描述，其中指出，离散系统的动态行为一般可以由差分方程描述。众所周知，离散系统的输入与输出仅在离散的时刻上取值，系统状态每隔固定的时间才更新一次；而 Simulink 对离散系统的仿真核心是对离散系统差分方程的求解。因此，Simulink 可以做到对离散系统仿真的绝对精确（除去有限的数据截断误差）。

在对纯粹的离散系统进行仿真时，需要选择离散求解器对其进行求解。用户只需选择 Simulink 仿真参数设置对话框中的求解器选项卡中的 discrete（no continuous states）选项，即没有连续状态的离散求解器，便可以对离散系统进行精确的求解与仿真。读者可以参考第 5 章中相关内容了解离散求解器的其它设置，这里不再赘述。

6.1.2 连续求解器

与离散系统不同，连续系统具有连续的输入与输出，并且系统中一般都存在着连续的状态变量。连续系统中存在的状态变量往往是系统中某些信号的微分或积分，因此连续系统一般由微分方程或与之等价的其它方式进行描述。这就决定了使用数字计算机不可能得到连续系统的精确解，而只能得到系统的数字解（即近似解）。

Simulink 在对连续系统进行求解仿真时，其核心是对系统微分或偏微分方程进行求解。因此，使用 Simulink 对连续系统进行求解仿真时所得到的结果均为近似解，只要此近似解在一定的误差范围之内便可。对微分方程的数字求解有不同的近似解法，因此 Simulink 的连续求解器有多种不同的形式，如变步长求解器 ode45、ode23、ode113，以及定步长求解器 ode5、ode4、ode3 等等。采用不同的连续求解器会对连续系统的仿真结果与仿真速度产生不同的影响，但一般不会对系统的性能分析产生较大的影响，因为用户可以设置具有一定的误差范围的连续求解器进行相应的控制。离散求解器与连续求解器设置的不同之处如图 6.1 所示。

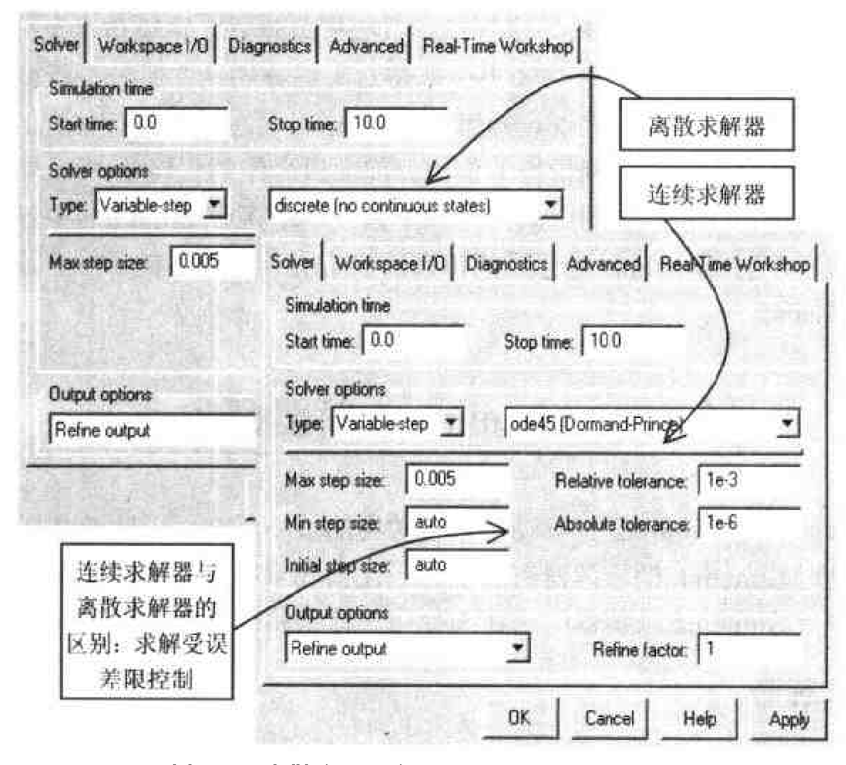


图 6.1 离散求解器与连续求解器设置的比较

为了使读者对 Simulink 的连续求解器有一个更为深刻的理解，在此对 Simulink 的误差控制与仿真步长计算进行简单的介绍。当然，对于定步长连续求解器，并不存在着误差控制的问题；只有采用变步长连续求解器，才会根据积分误差修改仿真步长。在对连续系统进行求解时，仿真步长计算受到绝对误差与相对误差的共同控制；系统会自动选用对系统求解影响最小的误差对步长计算进行控制。只有在求解误差满足相应的误差范围的情况下才可以对系统进行下一步仿真。

由于连续系统状态变量不能够被精确地计算出来，因而积分的误差值同样也是一个近似值。通常，连续求解器采用两个不同阶次的近似方法进行积分，然后计算它们之间的积分差值作为积分误差。连续求解器积分误差的计算如图 6.2 所示。

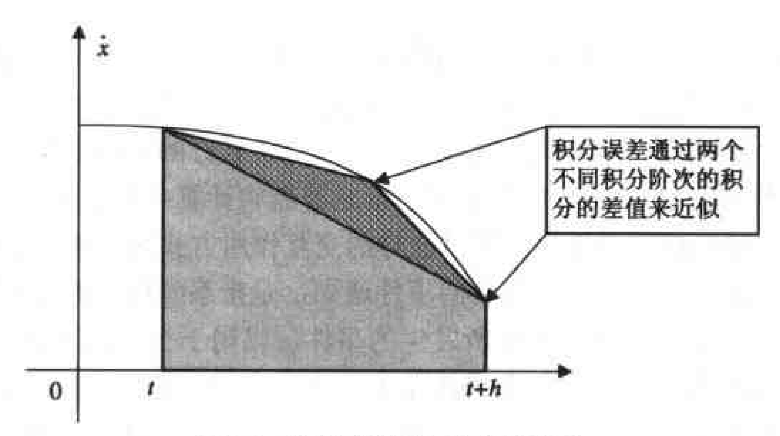


图 6.2 连续求解器积分误差计算

图 6.2 中 h 为积分步长。注意，此图以最简单的多边形积分近似算法为例说明积分误差的计算，在实际中具体的方法视连续求解器的不同而不同。如果积分误差满足绝对误差或相对误差，则仿真继续进行；如果不满足，则求解器尝试一个更小的步长，并重复这个过程。当然，连续求解器在选择更小步长时采用的方法也不尽相同。如果误差上限值的选择或连续求解器的选择不适合待求解的连续系统，则仿真步长有可能会变得非常小，使仿真速度变得非常慢。(用户需要注意这一点。)

对于实际的系统而言，很少有纯粹的离散系统或连续系统，大部分系统均为混合系统。连续变步长求解器不仅考虑了连续状态的求解，而且也考虑了系统中离散状态的求解。连续变步长求解器首先尝试使用最大步长（仿真起始时采用初始步长）进行求解，如果在这个仿真区间内有离散状态的更新，步长便减小到与离散状态的更新相吻合。

混合系统仿真时连续状态求解与离散状态求解的协调如图 6.3 所示。其中 h 为初始步长，由于在时刻 t 与 $t+h$ 之间系统存在着离散状态的更新，因而连续变步长求解器将会减小步长至 h_{new}，之后再计算积分误差以控制求解。如果求解误差满足误差范围，则进行下一步仿真，否则缩小时间间隔，重复此过程进行求解仿真。

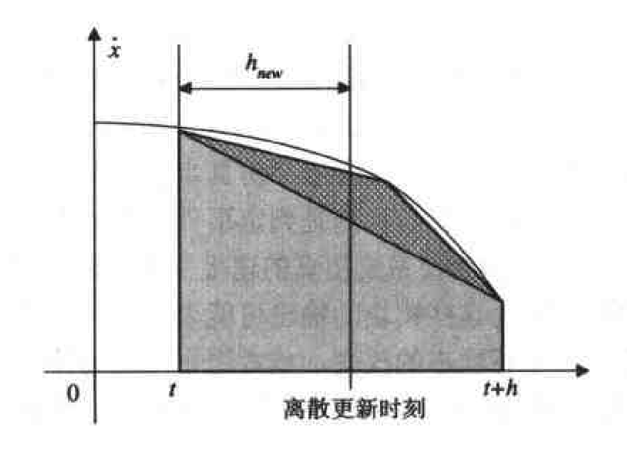


图 6.3 连续状态求解与离散状态求解的协调示意图

6.2 系统过零的概念与解决方案

6.1 节中对 Simulink 的求解器进行了较为深入的介绍。Simulink 求解器固然是系统仿真的核心，但 Simulink 对动态系统求解仿真的控制流程也是非常关键的。Simulink 对系统仿真的控制是通过系统模型与求解器之间建立对话的方式进行的：Simulink 将系统模型、模块参数与系统方程传递给 Simulink 的求解器，而求解器将计算出的系统状态与仿真时间通过 Simulink 环境传递给系统模型本身，通过这样的交互作用方式来完成动态系统的仿真。

对话方式的核心是事件通知。所谓的事件通知，是指系统模型通过 Simulink 仿真环境告知求解器在前一仿真步长内系统中所发生的事件，以用于当前仿真时刻求解器的计算。而过零检测则用来检测系统中是否有事件发生。系统模型正是通过过零检测与事件通知完成与 Simulink 求解器的交互。本节将详细介绍过零检测与事件通知的概念。

6.2.1 过零的产生

在动态系统的仿真过程中，所谓过零，是指系统模型中的信号或系统模块特征的某种改变。这种特征改变包括以下两种情况：

(1) 信号在上一个仿真时间步长之内改变了符号。

(2) 系统模块在上一个仿真时间步长改变了模式（如积分器进入了饱和区段）。

过零本身便是一个非常重要的事件，同时它也用来表示其它事件的发生。过零一般用来表征动态系统中的某种不连续性，例如系统响应中的跳变、输入信号的脉冲与阶跃等。如果在动态系统的仿真中不对过零进行检测，很可能会导致不准确的结果。这是因为对于某些系统而言，系统中的过零将会引起系统动态方程的改变，也就是动态系统的运行模式将发生变化。读者可以回忆在第 5 章动态系统仿真中所介绍的蹦极系统，在没有考虑过零事件发生时，系统的输出响应为一振荡的过程；但是此蹦极系统对于体重为 70 kg 的人来说是不安全的，跳跃者会触地身亡。如果考虑过零的发生（也就是跳跃者的触地），整个蹦极系统的动态响应将会发生改变，而非正常的振荡变化。

6.2.2 事件通知

在动态系统仿真中，采用变步长求解器可以使 Simulink 正确地检测到系统模块与信号中过零事件的发生。当一个模块通过 Simulink 仿真环境通知求解器，在系统前一仿真步长时间内发生了过零事件，变步长求解器就会缩小仿真步长，即使求解误差满足绝对误差和相对误差的上限要求。缩小仿真步长的目的是判定事件发生的准确时间（也就是过零事件发生的准确时刻）。当然，这样做会使系统仿真的速度变慢，但正如前面所说，这对于系统的某些模块是至关重要的。因为这些模块的输出可能表示了一个物理值，它的零值有着重要的意义；或者是标志系统运行状态的改变，或者用来控制另外一个模块等等。事实上，只有少量的模块能够发出事件通知。每个模块发出专属于自己的事件通知，而且可能与不止一个类型的事件发生关联。

事件通知是 Simulink 进行动态系统仿真的核心。可以这么说，Simulink 动态系统仿真

是基于事件驱动的，这符合当前交互式设计与面向对象设计的思想。在系统仿真中，系统模型与求解器均可以视为某种对象，事件通知相当于对象之间的消息传递；对象通过消息的传递来完成系统仿真的目的。系统模型与求解器之间的交互作用如图 6.4 所示。

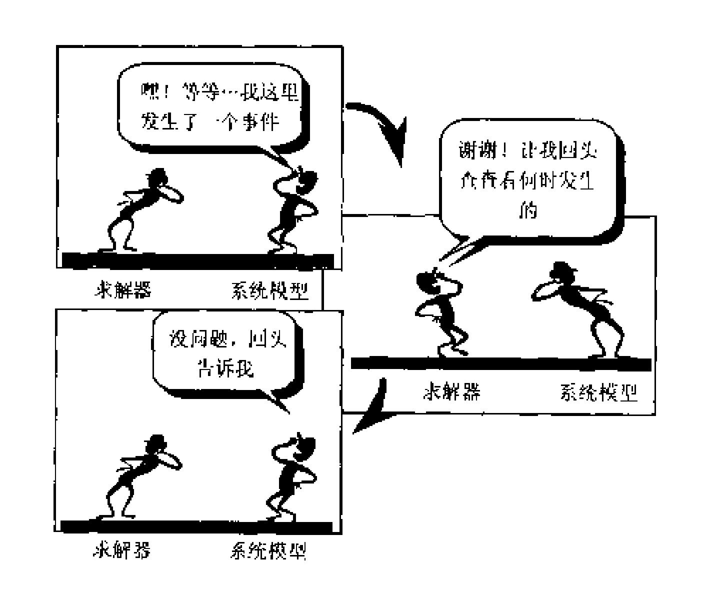


图 6.4　系统模型与求解器之间的交互作用示意图

6.2.3　支持过零的模块

在 Simulink 的模块库中，并非所有的模块都能够产生过零事件。能够产生过零事件的 Simulink 模块有：Abs（Math 数学库中的求取绝对值模块）、Backlash（Nonlinear 非线性库中的偏移模块）、Dead Zone（Nonlinear 非线性库中的死区模块）、Hit Crossing（Signals & Systems 信号与系统库中的零交叉模块）、Integrator（Continuous 连续库中的积分模块）、MinMax（Math 数学库中的最值模块）、Relational Operator（Math 数学库中的关系运算模块）、Relay（Nonlinear 非线性库中的延迟模块）、Saturation（Nonlinear 非线性库中的饱和模块）、Sign（Math 数学库中的符号运算模块）、Step（Sources 输入库中的阶跃模块）、Subsystem（Subsystems 子系统库中的子系统模块），以及 Switch（Nonlinear 非线性库中的开关模块）等。一般来说，不同模块所产生的过零的类型是有差异的。例如：对于 Abs 绝对值求取模块，当输入改变符号时产生一个过零事件，而 Saturation 饱和模块能够生成两个不同的过零事件，一个用于下饱和，一个用于上饱和。

对于其它的许多模块而言，它们不具有过零检测的能力。如果需要对这些模块进行过零检测，则可以使用信号与系统库（Signals & Systems）中的 Hit Crossing 零交叉模块来实现。当 Hit Crossing 模块的输入穿过某一偏移值（offset）时会产生一个过零事件，所以它可以用来为不带过零能力的模块提供过零检测的能力。

一般而言，系统模型中模块过零的作用有两种类型：一是用来通知求解器，系统的运行模式是否发生了改变，也就是系统的动态特性是否发生改变；二是驱动系统模型中其它模块。过零信号包含三种类型：上升沿、下降沿、双边沿，如图 6.5 所示。

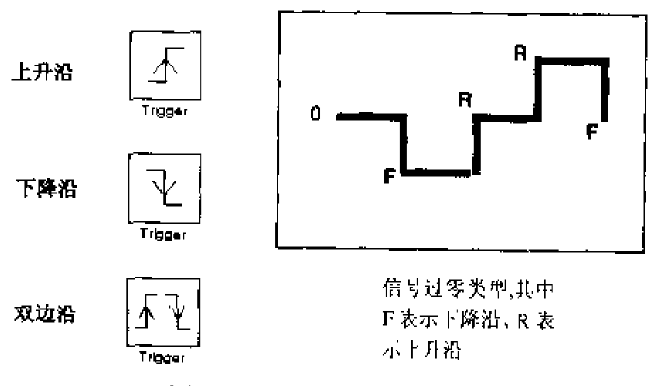


图 6.5　过零信号的类型

下面分别对这三种类型进行简单的介绍。

(1) 上升沿：系统中的信号上升到零或穿过零，或者是信号由零变为正。

(2) 下降沿：系统中的信号下降到零或穿过零，或者是信号由零变为负。

(3) 双边沿：任何信号的上升或下降沿的发生。

6.2.4 过零的举例——过零的产生与关闭过零

1. 过零点的产生

【例 6.1】 过零的产生与影响。

这里以一个很简单的例子来说明系统中过零的概念以及它对系统仿真所造成的影响。在这个例子中，采用 Functions & Tables-函数与表库中的 Function 函数模块和 Math 数学库中的 Abs 绝对值模块分别计算对应输入的绝对值。我们知道，Function 模块不会产生过零事件，所以在求取绝对值时，一些拐角点被漏掉了；但是 Abs 模块能够产生过零事件，所以每当它的输入信号改变符号时，它都能够精确地得到零点结果。图 6.6 所示为此系统的 Simulink 模型以及系统仿真结果。

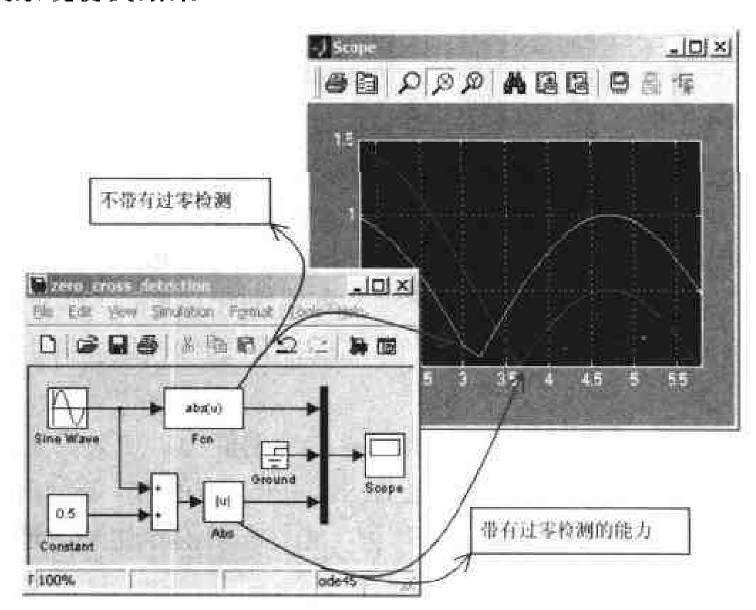


图 6.6 过零产生的影响

从仿真的结果中可以明显地看出，对于不带有过零检测的 Function 函数模块，在求取输入信号的绝对值时，漏掉了信号的过零点（即结果中的拐角点）；而对于具有过零检测能力的 Abs 求取绝对值模块，它可以使仿真在过零点处的仿真步长足够小，从而可以获得精确的结果。为说明这一点，在 MATLAB 命令窗口中输入如下语句：

```
>> semilogy(tout(1:end-1,diff(tout))
% 绘制系统仿真时刻的一阶差分（即系统仿真步长），如图 6.7 所示，其中常规步长为 0.2 s，
% 当发生过零的情况时，系统仿真步长自动缩小至约 10^-14 s
```

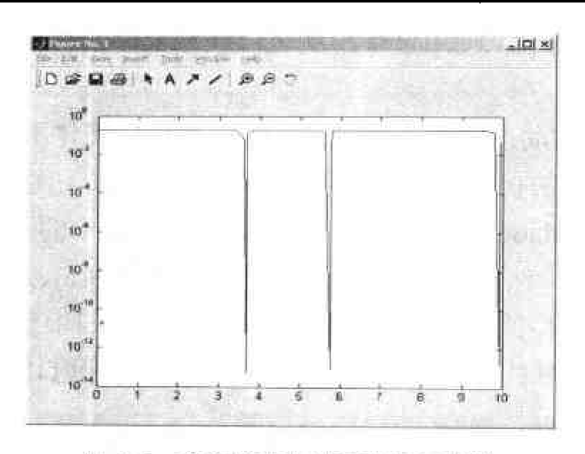
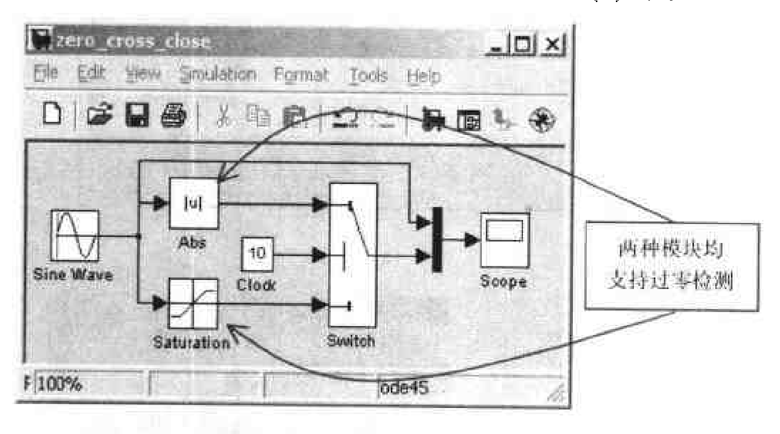


图 6.7　系统仿真中过零处步长变化

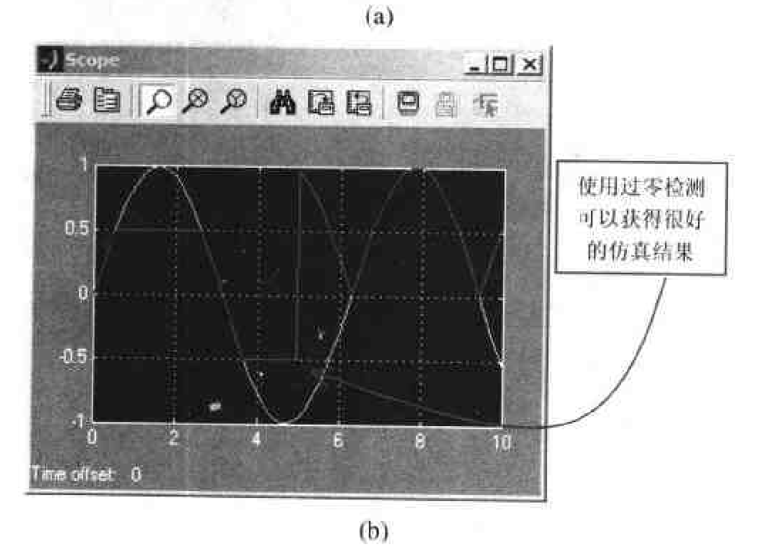
2. 关闭过零

【例 6.2】　过零的关闭与影响。

在【例 6.1】中，过零表示系统中信号穿过了零点。其实，过零不仅用来表示信号穿过了零点，还可以用来表示信号的陡沿和饱和。在这个例子中，系统实现了输入信号由其绝对值跳变到饱和值的功能，而且跳变过程受到仿真时刻的控制。在此系统模型中所使用的 Abs 模块与 Saturation 模块都支持过零事件的产生，因此在系统的响应输出中得到了理想的陡沿。其中系统模型如图 6.8(a)所示，系统仿真结果如图 6.8(b)所示。



(a)

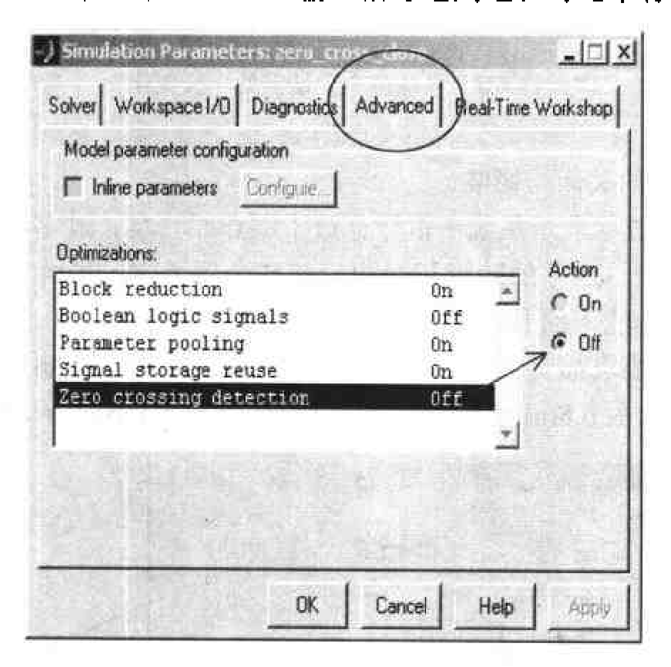


(b)

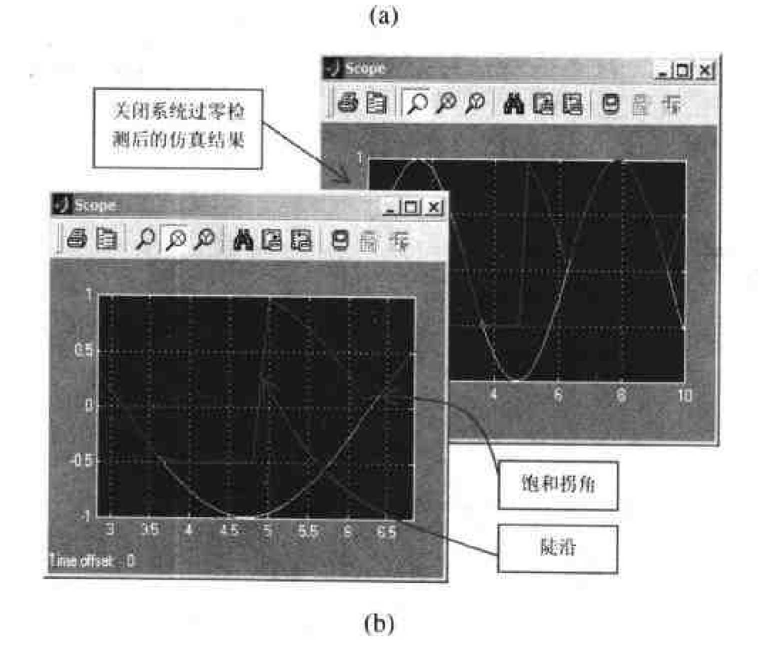
图 6.8　系统模型及系统仿真结果

从图 6.8 中可以明显看出，使用过零检测可以获得很好的仿真结果，系统的输出具有很好的陡沿。

在使用 Simulink 进行动态系统仿真中，其默认参数选择使用过零检测的功能。如果使用过零检测并不能给系统的仿真带来很大的好处，用户可以关闭仿真过程中过零事件的检测功能。用户可以在 Simulation Parameters 参数设置对话框中的 Advanced 选项卡中进行设置，以关闭过零检测功能，然后再次对系统进行仿真。图 6.9(a)、(b)所示分别为关闭过零检测的设置以及在关闭过零检测后系统的仿真结果。显然，关闭过零检测功能使得仿真结果在 t=5 时不是非常理想，而且当 Saturation 输入信号达到饱和时还带有一些拐角。



(a)



(b)

图 6.9 关闭系统过零检测的设置和关闭过零检测后的仿真结果

6.2.5　使用过零检测的其它注意事项

在使用过零检测时，用户需要注意如下几点：

(1) 关闭系统仿真参数设置中的过零事件检测，可以使动态系统的仿真速度得到很大的提高。但可能会引起系统仿真结果的不精确，甚至出现错误结果。

(2) 关闭系统过零检测对 Hit Crossing 零交叉模块并无影响。

(3) 对于离散模块及其产生的离散信号不需要进行过零检测。这是因为用于离散系统仿真的离散求解器与连续变步长求解器都可以很好地匹配离散信号的更新时刻。

此外，对于某些比较特殊的动态系统而言，在对其进行仿真时，有可能在一个非常小的区间内多次通过零点。这将导致在同一时间内多次探测到信号的过零，从而使得 Simulink 仿真终止。在这种情况下，用户应该在仿真参数设置中关闭过零检测功能。当然，这些系统通常是某些物理现象的理想模型，如无质量弹簧的振荡、没有任何延迟的气压系统等。但是对于某些系统而言，这些模块的过零非常重要，此时用户可以采用在系统模型中串入零交叉 Hit Crossing 模块，并关闭仿真过零检测功能来实现过零的使用。

6.3　系统代数环的概念与解决方案

6.3.1　直接馈通模块

在使用 Simulink 的模块库建立动态系统的模型时，有些系统模块的输入端口（Input ports）具有直接馈通（Direct feedthrough）的特性。所谓模块的直接馈通，是指如果在这些模块的输入端口中没有输入信号，则无法计算此模块的输出信号。换句话说，直接馈通就是模块输出直接依赖于模块的输入。在 Simulink 中具有直接馈通特性的模块有如下的几种：

(1) Math Function 数学函数模块。

(2) Gain 增益模块。

(3) Product 乘法模块。

(4) State-Space 状态空间模块（其中矩阵 $\boldsymbol{D}$ 不为 0）。

(5) Transfer Fcn 传递函数模块（分子与分母多项式阶次相同）。

(6) Sum 求和模块。

(7) Zero-Pole 零极点模块（零点与极点数目相同）。

(8) Integrator 积分模块。

6.3.2　代数环的产生

在介绍完具有直接馈通特性的系统模块之后，来介绍代数环的产生。系统模型中产生代数环的条件如下：

(1) 具有直接馈通特性的系统模块的输入，直接由此模块的输出来驱动。

(2) 具有直接馈通特性的系统模块的输入，由其它直接馈通模块所构成的反馈回路间接来驱动。

图 6.10 所示为一个非常简单的标量代数环的构成。

此代数环回路仅由一个求和模块构成，其中模块的输出状态 z 同时作为此模块的输入。由于求和模块具有直接馈通的特性，也就是模块输出直接依赖于模块的输入，因而构成了代数环。不难看出此代数环可以用如下的数学表达式来描述：

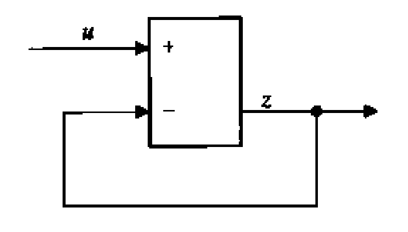


图 6.10 标量代数环

$$z = u - z$$

显然，此系统模块的输出状态为 $z = u/2$。但是对于大多数的代数环系统而言，难以通过直接观察来求解。

如果系统模型中出现了代数环，由于代数环的输入输出之间是相互依赖的，组成代数环的所有模块都要求在同一个时刻计算输出，这与系统仿真的顺序概念相反，因此，最好使用其它的方法（如手工的方法）对系统方程进行求解、对代数环进行代数约束或切断环来解决代数环的求解问题。

6.3.3 代数环的举例与解决方案之一：直接求解系统方程

【例 6.3】 代数环的直接求解。在图 6.11 中所示的两个系统模型中均存在代数环结构，试对这两个系统进行求解。

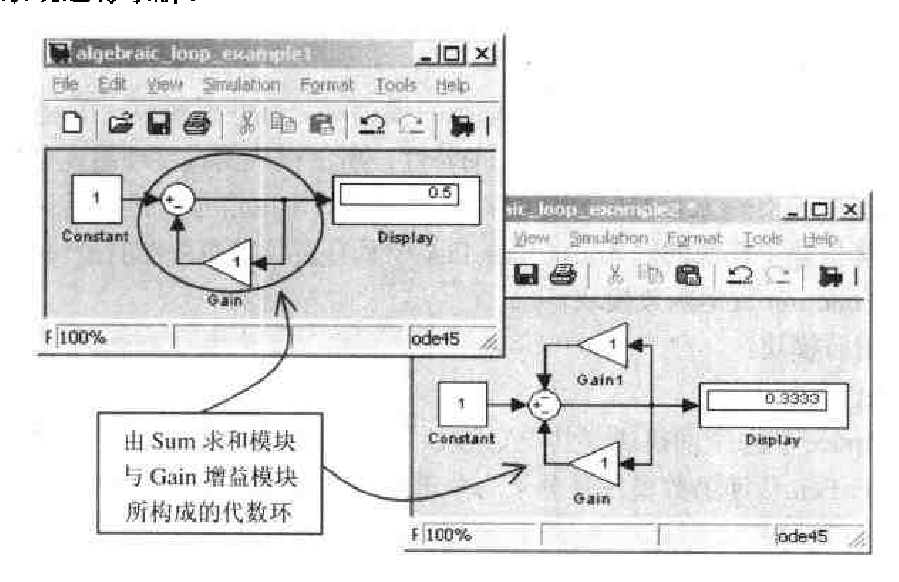


图 6.11 具有代数环的系统模型

解：为了计算求和模块 Sum 的输出，需要知道其输入，但是其输入恰恰包含模块的输出。对于此二系统，很容易写出如下所示的系统的动态方程：

(1) 第一个系统模型的动态方程：$z = 1 - z$，所以 $z = 0.5$。

(2) 第二个系统模型的动态方程：$z = 1 - z - z$，所以 $z = 0.3333$。

其结果如图 6.11 中 Display 模块所示。其实在 Simulink 中有一个内置的代数环求解器，可以对诸如上述的系统模型进行正确的计算。并且 Simulink 会在 MATLAB 命令窗口中给出仿真警告，告诉用户系统模型中存在的代数环情况。例如，在对第二个系统模型进行仿真时，MATLAB 窗口中显示如下警告信息：

```
Warning: Block diagram 'algebraic_loop_example2' contains 1 algebraic loop(s).
Found algebraic loop containing block(s):
   'algebraic_loop_example2/Gain1'
   'algebraic_loop_example2/Gain'
   'algebraic_loop_example2/Sum' (algebraic variable)
```

这意味着此系统模型中存在着一个代数环，构成代数环的组成模块是 Gain1、Gain 以及 Sum 模块。

6.3.4　代数环的举例与解决方案之二：代数约束

用户除了可以使用 Simulink 内置的代数环求解器对含有代数环的动态系统进行仿真，还可以使用 Math 模块库中的代数约束 Algebraic Constraint 模块对动态系统数学方程进行求解。

使用代数约束模块并给出约束初始值，可以方便地对代数方程进行求解。代数约束模块通过调整其输出代数状态以使其输入 $F(z)$ 为零。其中 z 为模块的输出状态，$F(z)$ 为一代数表达式，它作为模块的输入。注意，代数约束模块的输出必须通过一个反馈回路以影响模块的输入，而且必须给定一个模块输出状态的初始值以有效地改进代数环的效率。

【例 6.4】确良 使用代数约束求解代数环。

在如图 6.12 所示的系统模型中代数约束模块的输出分别为代数状态 z1, z2 。z1, z2 分别通过反馈回路作为代数约束模块的输入。运行此系统相当于对如下的代数方程进行求解：

$$\begin{cases} z1+z2-1=0 \\ z2-z1-1=0 \end{cases}$$

其仿真结果如图 6.12 中 Display 模块所显示的那样，其中 $z1=0$ ，$z2=1$ 。

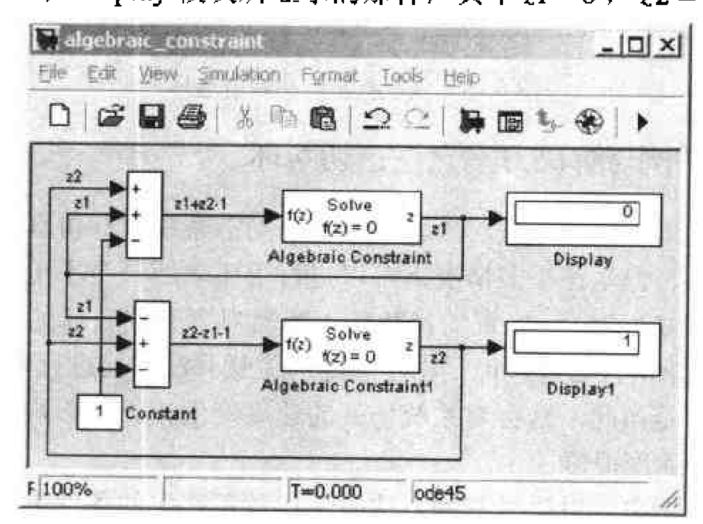


图 6.12　使用代数约束求解的代数环结构

当在系统模型中使用代数约束时，系统中将出现代数环。对于系统模型中包含代数环结构的系统而言，Simulink 会在每一仿真步长内调用代数环求解器对系统进行求解。代数环求解器通过迭代的方法对系统进行求解，由于对系统的求解使用了迭代的方法，因而对含有代数环的系统仿真速度要慢于一般不含代数环的系统。

在使用代数约束模块时，Simulink 使用牛顿迭代法求解代数环。虽然采用这种方法是一种稳定的算法，但是如果代数状态的初始值选择得不合适，则算法有可能不收敛。因此，用户在使用代数约束时，一定要注意对代数约束模块输出的代数状态的初始值的选取问题，如果初始值选取得不同，有可能造成最终结果的不同。

【例 6.5】代数状态的初始值选取。使用代数约束来求解方程

$$x^2 - x - 2 = 0 \quad \text{即} (x+1)(x-2) = 0$$

的根（显然此方程的根为 $x_1 = -1$ 及 $x_2 = 2$）。

解：首先建立如图 6.13 所示的系统模型，然后对代数约束模块的初始值进行设置，如图 6.13 所示（仿真结果如 Display 模块中所示）。

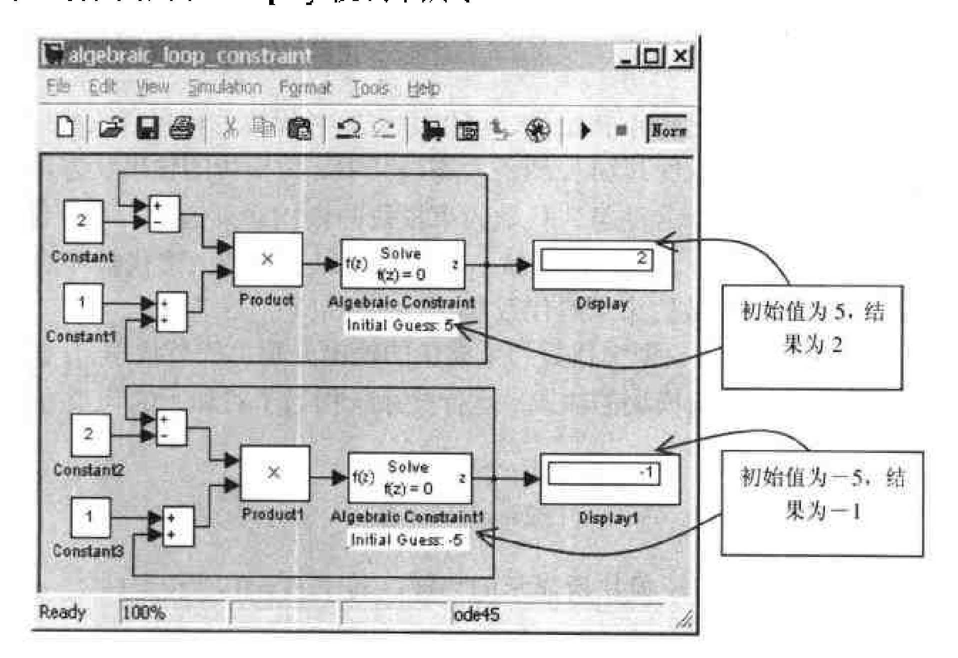


图 6.13 代数状态的初始值选择

6.3.5 代数环的举例与解决方案之三：切断环

至此，读者能够采用两种方法对含有代数环的动态系统进行仿真分析：一是直接对系统方程进行手工求解，但是在很多情况下难以进行手工求解甚至不可能进行手工求解；二是使用代数约束，由 Simulink 内置的代数环求解器对含有代数环的系统进行仿真。虽然 Simulink 能够有效地求解代数环，但是由于采用的牛顿算法需要进行多次迭代以求解系统，而且在每一时间步都是如此，这会对系统仿真的速度产生较大的影响，从而导致了含有代数环结构的系统仿真速度很慢。

在有些情况下，用户可以通过某种方法破坏代数环产生的条件来“切断”系统模型中的代数环结构，从而加快系统仿真的速度。一个最常用的技巧是通过加入一个存储模块（Continuous 模块库中的 Memory 模块）或延迟单元（Discrete 模块库中的 Unit Delay 模块）以切断代数环。尽管使用这个方法非常容易，但是在一般条件下并不推荐这样做，因为加入存储模块会改变系统的动态性能，而且对于不适当的初始估计值，有可能导致系统不稳定。

【例 6.6】 对于如下的连续线性系统：

$$G(s)=\frac{s+0.5}{(s-2)(s+3)}$$

建立如图 6.14 所示的系统模型。

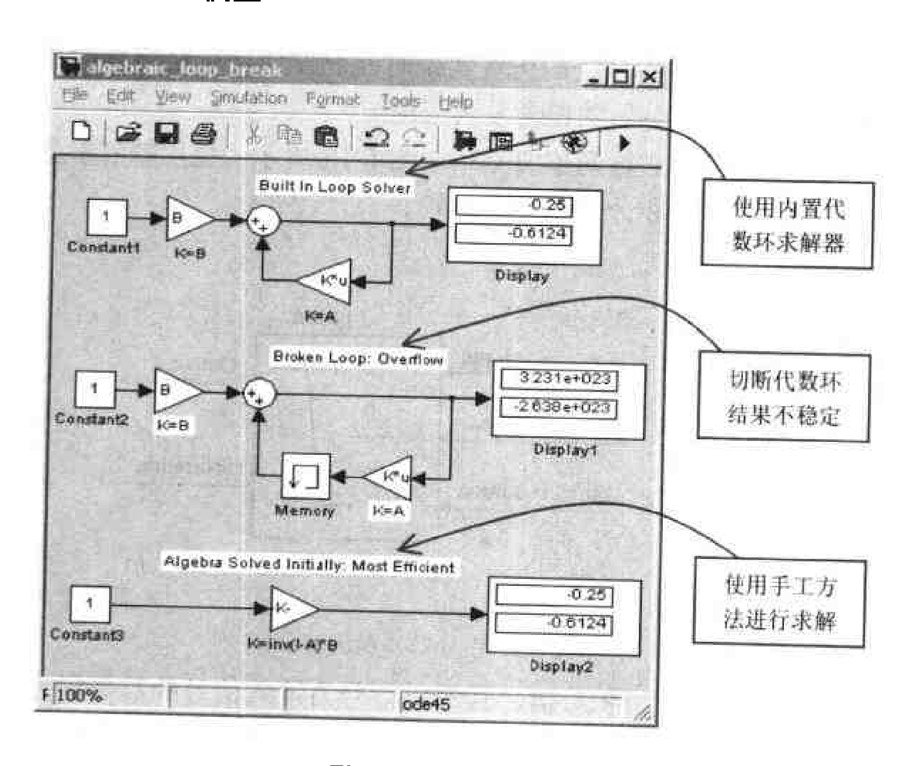


图 6.14　切断代数环

图 6.14 中 $G(s)$ 为线性连续系统的零极点描述，其相应的状态空间描述矩阵分别为 ***A***，***B***，***C***，***D***。建立此系统的模型以求出系统在恒定输入下的状态值。为了说明切断代数环的影响，我们在此模型中给出系统状态求取的三种不同方法以作为比较。图中最上方为使用 Simulink 的内置代数环求解器进行状态求解，最下方为使用手工方式进行状态求解，中间为使用 Memory 模块切断代数环，然后进行状态求解。

对此系统进行仿真之前，由于使用了此线性连续系统的状态空间描述矩阵变量作为系统模块的参数，因此必须保证 MATLAB 工作空间中这些矩阵变量的存在。读者可以在 MATLAB 的命令窗口中键入如下的语句以求取系统状态空间描述矩阵：

```
>> [A,B,C,D]=zp2ss([0.5],[2,-3],1);
>> I=eye(2);                          % 并生成单位矩阵 I
```

然后运行此系统进行系统状态的求解。图中在 Display 模块中给出了系统的运行结果，从中可以明显看出，无论是使用 Simulink 的内置代数环求解器还是使用手工计算的方法，都可以获得正确的系统状态值；而使用 Memory 模块切断代数环之后，导致了系统的不稳定，从而使得状态值的求解发散。由此可以看出，虽然使用 Memory 模块可以切断代数环，但是它却改变了系统的动态特性，因此在一般的情况下，用户应该尽量不要切断代数环。如果用户使用 Memory 模块切断代数环，一定要检测 Memory 模块所需的初始状态是否能够使系统收敛，如果系统不收敛，则使用 Memory 会使系统运行失败。

6.4 高级积分器

在使用 Simulink 对实际的动态系统进行仿真时，积分运算可以说是 Simulink 求解器的核心技术之一。前面在介绍动态系统的仿真实现时，仅仅使用了最简单的积分器设置。在这一节中，将简单介绍高级积分器的概念及其应用。

首先对积分器的各个端口进行简单的介绍。图 6.15 所示为使用缺省参数设置下的积分器外观与选择所有参数设置后积分器的外观比较。

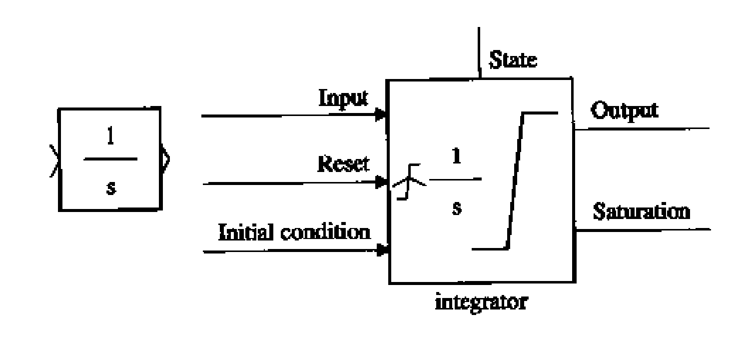


图 6.15 积分器外观比较

对于使用缺省参数设置下的积分器，其输出信号为输入信号的数值积分，想必读者对其已经比较熟悉了，这里不再赘述。下面详细介绍一下选择所有参数设置后的积分器各个端口的含义以及对积分器的设置。首先介绍一下积分器参数设置对话框，如图 6.16 所示。

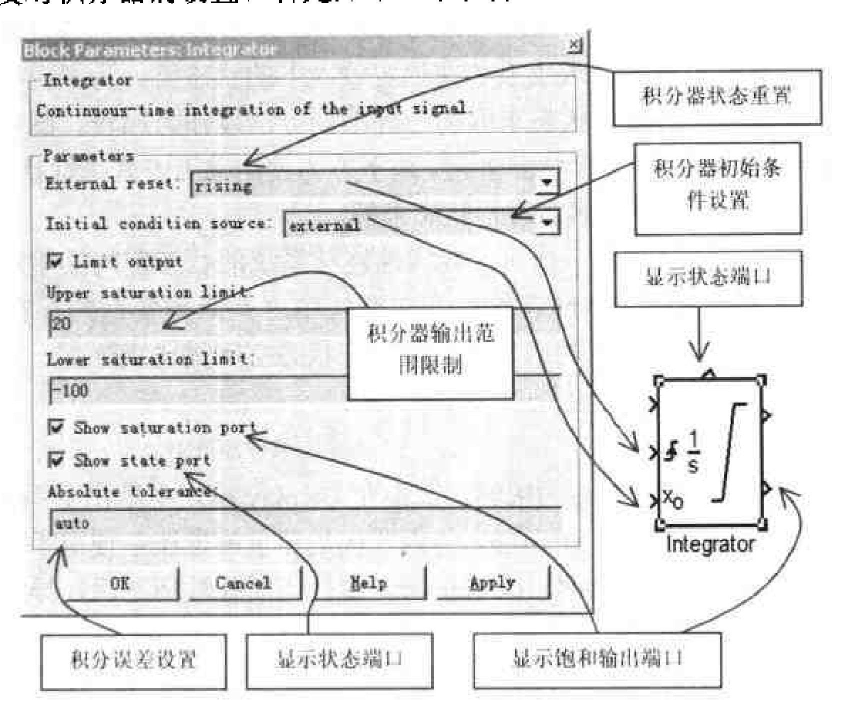


图 6.16 高级积分器设置

1. 积分器的初始条件端口（Initial condition）

设置积分器初始条件的方法有两种，它们分别是：

(1) 在积分器模块参数设置对话框中设置初始条件：在初始条件源设置（Initial condition source）中选择内部设置（Internal），并在下面的文本框中键入给定的初始条件，此时不显示积分器端口。

(2) 从外部输入源设置积分器初始条件：在初始条件源设置中选择外部设置（External），初始条件设置端口以 x_0 作为标志。此时需要使用 Signals & Systems 模块库中的 IC 模块设置积分器初始值。

2. 积分器状态端口（State）

当出现下述的两种情况时，需要使用积分器的状态端口而非其输出端口：

(1) 当积分器模块的输出经重置端口或初始条件端口反馈至模块本身时，会造成系统模型中出现代数环结构的问题，此时需要使用状态端口。

(2) 当从一个条件执行子系统向另外的条件执行子系统传递状态时，可能会引起时间同步问题。此时也需要使用状态端口而非输出端口。至于条件执行子系统的有关内容，将在第 7 章中介绍。

其实状态端口的输出值与输出端口的输出值本身并没有区别，其不同之处在于二者产生的时间略微有所不同（都处于同样的时间步之内），这正是 Simulink 避免出现上述问题的解决方案。选择 Show state port 复选框，状态端口将显示在积分器的顶部。

3. 积分器输出范围限制与饱和输出端口（Saturation）

在某些情况下，积分器的输出可能会超过系统本身所允许的上限或下限值，选择积分器输出范围限制框（Limit output），并设置上限值（Upper saturation limit）与下限值（Lower saturation limit），可以将积分器的输出限制在一个给定的范围之内。此时积分器的输出服从下面的规则：

(1) 当积分结果小于或等于下限值并且输入信号为负，积分器输出保持在下限值（下饱和区）。

(2) 当积分结果在上限值与下限值之间时，积分器输出为实际的积分结果。

(3) 当积分结果大于或等于上限值并且输入信号为正，积分器输出保持在上限值（上饱和区）。

选择 Show saturation port 复选框可以在积分器中显示饱和端口，此端口位于输出端口的下方。饱和端口的输出取值有三种情况，用来表示积分器的饱和状态：

(1) 输出为 1，表示积分器处于上饱和区。

(2) 输出为 0，表示积分器处于正常范围之内。

(3) 输出为-1，表示积分器处于下饱和区。

当选择输出范围限制时，积分器模块将产生三个过零事件：一个用来检测积分结果何时进入上饱和区，一个用来检测积分结果何时进入下饱和区，还有一个用来检测积分器何时离开饱和区。

4. 积分器重置

选择积分器状态重置框可以重新设置积分器的状态，其值由外部输入信号决定。此时，在积分器输入端口下方出现重置触发端口。可以采用不同的触发方式对积分器状态进行

重置：

(1) 当重置信号具有上升沿（rising）时，触发重置方式选择为上升沿。

(2) 当重置信号具有下降沿（falling）时，触发重置方式选择为下降沿。

(3) 当重置信号具有上升或下降沿（即双边沿）时，触发重置方式可选择为 either。

(4) 当重置信号非零时，选择 level 重置积分器状态，并使积分器输出保持在初始状态。

积分器的重置端口具有直接馈通的特性。积分器模块的输出，无论是直接反馈还是通过具体直接馈通特性的模块反馈至其重置端口，都会使系统模型中出现代数环结构。而使用状态端口代替输出端口可以避免代数环的出现。

5. 积分器绝对误差设置（Absolute tolerance）

在默认情况下，积分器采用 Simulink 自动设置的绝对误差限。用户也可以根据自己的需要设置积分器的绝对误差限，在 Absolute tolerance 下键入误差上限即可。

至此，我们对高级积分器设置做了一个比较全面的介绍，下面举例说明。

【例 6.7】 高级积分器的使用。系统模型如图 6.17 所示。

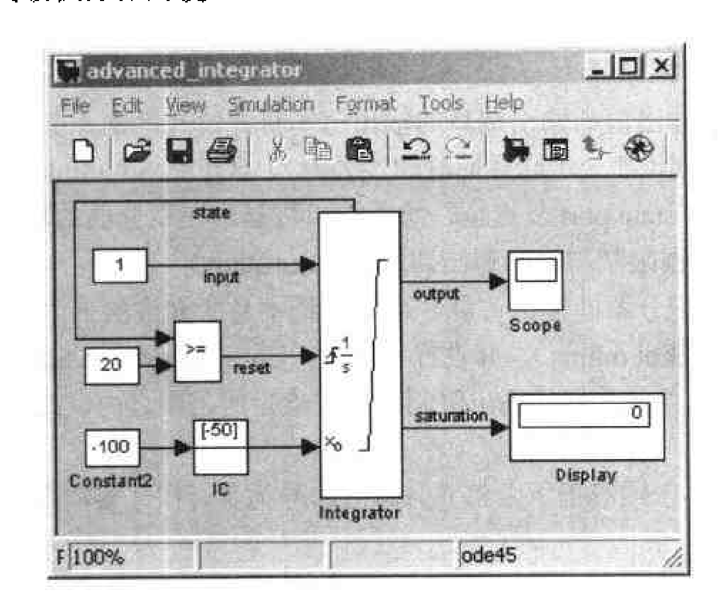


图 6.17 高级积分器的使用

在此系统中，积分器模块的输出（其实是积分器状态）驱动自身的状态重置端口。采用状态端口驱动的目的在于：使用它可以避免使用输出端口驱动重置状态所造成的代数环。这是因为，状态值（与输出值相等）的计算先于输出值，所以使用状态值驱动可以在输出之前判断是否需要对积分器进行重置。积分器初始时刻的初始值由 IC 模块提供，其值为-50。当积分器输出大于或等于 20 时，它将重置为由常数模块所给出的初始值 100。积分器的输入为一常值信号 1。

设置合适的仿真参数如下：仿真时间范围为 0~200，积分器的输出上限为 20，下限为 −100，状态重置选择信号上升沿 rising，其余参数如系统模型框图中所示。运行仿真结果如图 6.18 所示。从仿真结果中可以看出，在系统运行的初始时刻，积分器状态由 IC 模块所决定；而当系统的输出大于或等于 20 时，积分器状态重置为 −100。

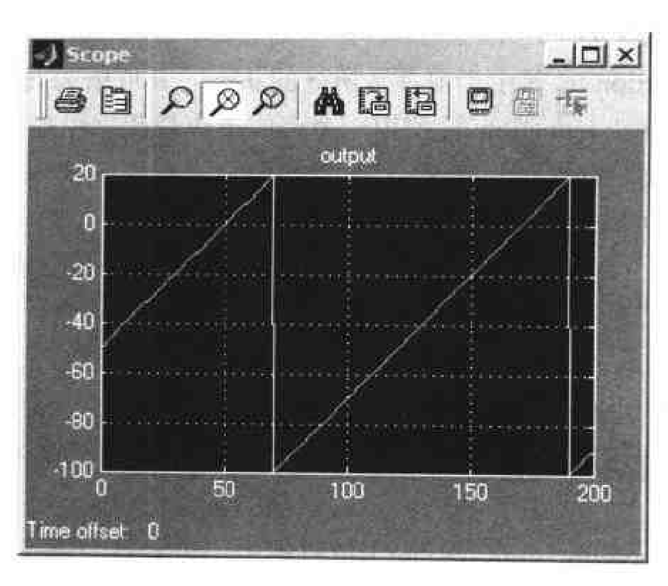


图 6.18　系统仿真结果

6.5　仿真参数设置：高级选项与诊断选项

在第 5 章中讲述 Simulink 动态系统仿真时，其中介绍了 Simulink 仿真参数选项的大部分普通设置，使用这些设置对于一般的系统仿真已经足够了。但是在某些情况下，用户需要对系统仿真进行控制以满足特殊的要求。Simulink 仿真参数的高级选项（Advanced）与诊断选项（Diagnostics），给用户提供了优化系统仿真的能力。用户可以根据需要对 Simulink 的高级选项进行设置以满足仿真的要求。本节将简单介绍 Simulink 仿真参数的高级设置：高级选项与诊断选项。

6.5.1　高级选项

首先介绍 Simulink 仿真参数的高级选项，如图 6.19 所示。

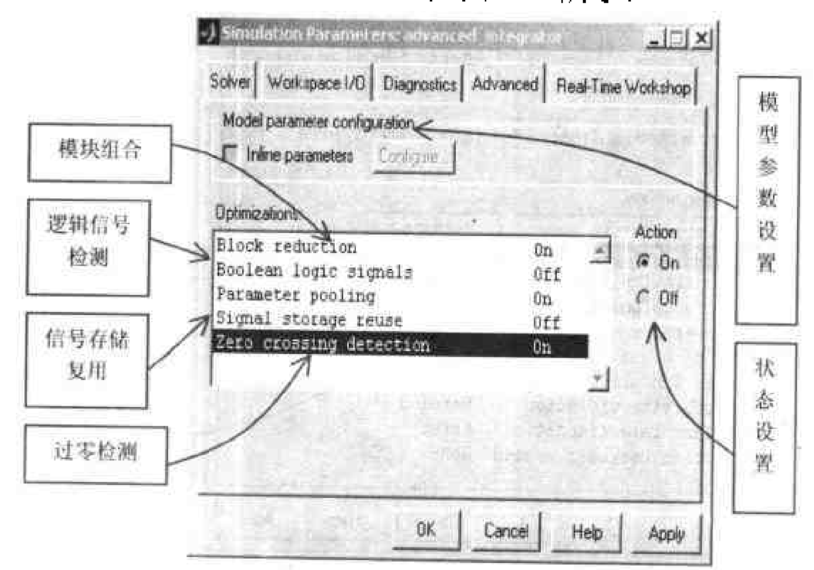


图 6.19　仿真参数的高级选项卡

其中各个选项的意义如下：

(1) 状态设置（Action）：显示当前参数所处的状态，其取值为开（on）或关（off）。

(2) 模块组合（Block reduction）：使用一个合成模块替代一组模块。

(3) 逻辑信号检测（Boolean logic signals）：检测逻辑模块的输入是否为 0。激活此状态（Action 设置为 on）可以强制模块输入为逻辑值。

(4) 信号存储复用（Signal storage reuse）：复用内存空间以节省信号所使用的内存。

注意：在系统使用悬浮显示时应该关闭此选项。

(5) 过零检测（Zero crossing detection）：检测过零事件的发生。

(6) 模型参数设置（Model parameter configuration）：设置整个系统模型的参数，此参数仅影响到在执行过程中可以改变系统参数的仿真或模型本身的参数可以由其它模型访问的系统模型产生的代码。

此外，Parameter pooling 选项在代码生成时使用，仿真时不需要改变它。

6.5.2 诊断选项

在对动态系统进行仿真时，很难避免一些问题的出现，如系统模型中存在的代数环结构、系统中数据连续的转换以及信号的越界等等。使用 Simulink 仿真参数对话框中的诊断选项可以对动态系统仿真过程中出现的问题进行诊断，同时也可以在系统仿真之前进行特定的测试，以有效地提高系统仿真的性能。需要注意的是，在诊断选项中的一致性与边界测试在许多情况下不是必需的。由于这些测试非常耗时，因此在一般的情况下不需要对其进行测试（选择这两个选项中的 None 选项即可）；但是如果系统模型中包含用户编写的模块，为避免仿真性能与内存使用方面的问题，应该设置适当的选项（警告 Warning 或错误 Error）。诊断选项对话框还包含了其它诸多潜在的问题，这些问题并不影响仿真计算的精度；因此可按照用户的要求，选择在出现这些问题时所要采取的动作（不予理睬 None，显示警告 Warning，终止并显示错误信息 Error）。这些问题都比较易于理解，这里就不再详述。图 6.20 所示为诊断选项卡设置。

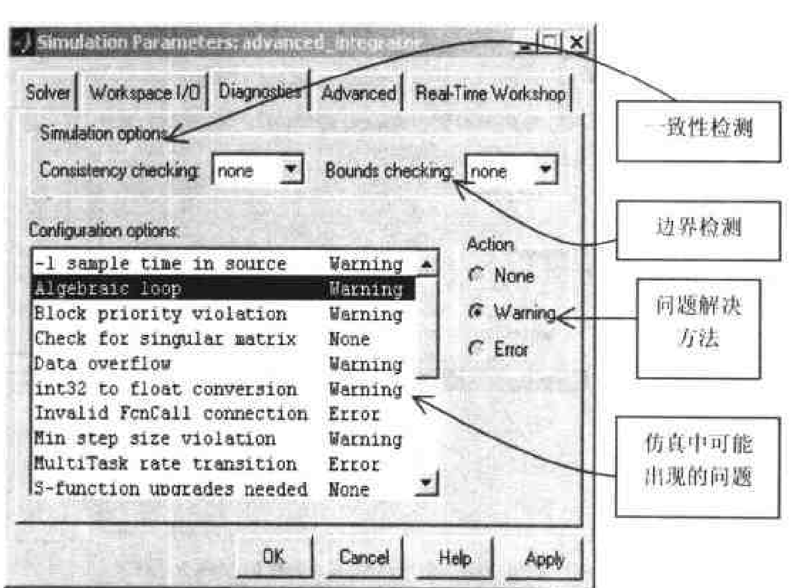
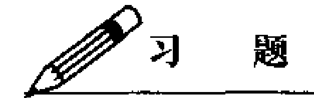


图 6.20　仿真参数之诊断选项卡



习　题

1. 一个混合系统中包含连续状态与离散状态，而且离散信号的采样时间各不相同。在系统仿真时应该使用何种 Simulink 求解器？

提示：连续变步长求解器。

2. 按照【例 6.1】与【例 6.2】，请读者自己建立系统模型并运行仿真，观察过零检测对系统仿真性能的影响。

3. 使用代数约束求解方程：

$$x^2+3x-4=0$$

提示：参考【例 6.5】。

4. 实现【例 6.7】中的系统，练习使用高级积分器。

第7章 Simulink 子系统技术

内容概要

· Simulink 的高级子系统技术
· Simulink 子系统封装技术
· Simulink 模块库技术

在使用 Simulink 建立动态系统的模型并进行系统仿真分析时，对于简单的动态系统仿真，用户可以直接建立其模型，然后进行动态仿真。然而对于复杂的动态系统而言，直接建立系统并仿真会给用户带来诸多不便，用户需要采用某种合适的策略建立系统模型，然后进行系统仿真。这是因为对于复杂的动态系统而言，系统中包含的功能模块较多，它们之间的输入输出关系比较复杂。这无疑会给用户建立系统模型、设置参数进行系统仿真以及对系统进行分析带来很大的不便，尤其对于系统的调试与诊断，当系统中出现问题时，难以对其进行分析与定位。在建立复杂系统的模型与系统仿真中，一般使用以下两种策略：

(1) 自下向上的策略：首先建立复杂系统中的每一个功能模块，然后再组合这些模块，逐渐建立整个系统模型。

(2) 自上向下的策略：首先建立复杂系统的整体结构模型，然后再实现系统每一部分的模型。

建立系统模型之后，便可以对其进行仿真分析了。无论采用何种策略建立复杂系统模型并进行仿真，其中都会不同程度使用 Simulink 的子系统技术。第 4 章中对 Simulink 子系统的概念与其应用作了简单的介绍。鉴于子系统技术在复杂系统的建模、仿真与调试诊断之中的广泛应用，本章将在第 4 章的基础之上进一步介绍 Simulink 更高级的子系统的概念、子系统模块封装，以及 Simulink 的模块库的生成与使用方法。

7.1 Simulink 简单子系统概念：回顾与复习

在介绍 Simulink 高级子系统的概念与使用之前，本节首先对第 4 章所介绍的 Simulink 通用子系统的概念与使用作一个简单的回顾与复习。随后的几节内容将对 Simulink 的高级子系统技术、子系统的封装与 Simulink 模块库的生成作详细且全面的介绍，从而使用户能够领略到 Simulink 的强大功能。

7.1.1 通用子系统的生成

在使用 Simulink 子系统技术时，通常子系统的生成有如下两种方法：

(1) 在已经建立好的系统模型之中建立子系统（如图 7.1 所示）。首先选择能够完成一定功能的一组模块，然后选择 Simulink 模型创建编辑器中 Edit 菜单下的 Create Subsystem，即可建立子系统并将这些模块封装到此子系统中，Simulink 自动生成子系统的输入与输出端口。

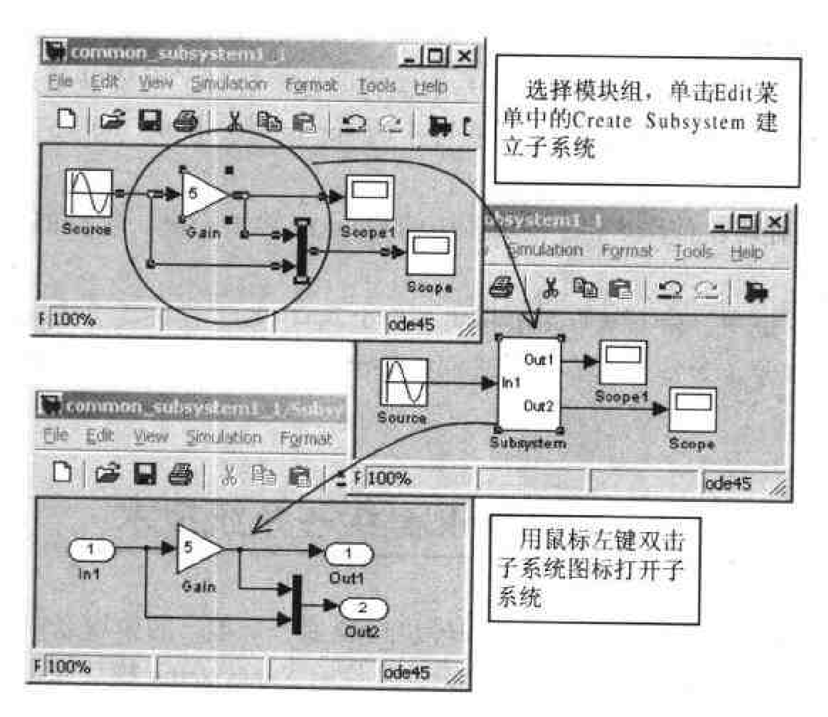


图 7.1　子系统建立：由模块组生成子系统

(2) 在建立系统模型时建立空的子系统（如图 7.2 所示）。使用 Subsystems 模块库中的 Subsystem 模块建立子系统，首先构成系统的整体模型，然后编辑空的子系统内的模块。注

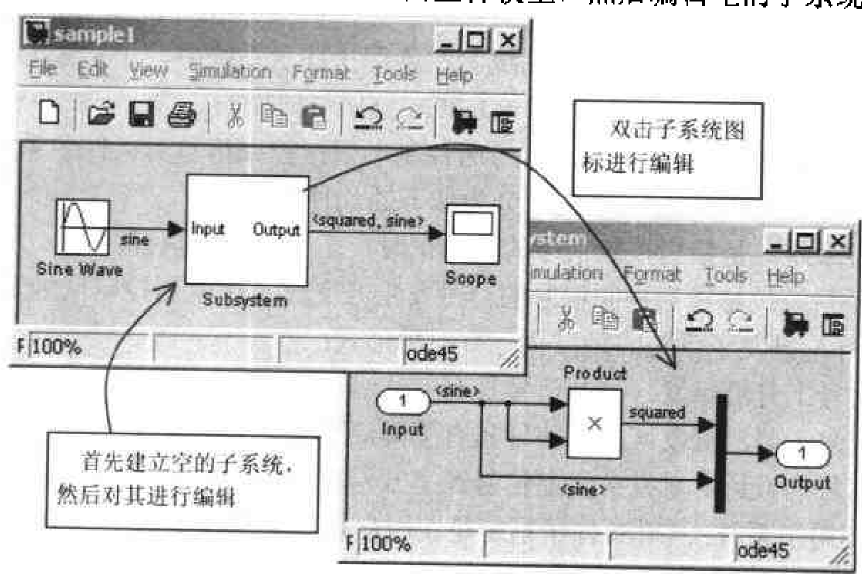


图 7.2　子系统建立：生成空子系统并编辑

意，对于多输入与多输出子系统而言，需要使用 Sources 模块库中的 In1 输入虚模块与 Sinks 模块库中的 Out1 输出虚模块来实现。

7.1.2 子系统的基本操作

在用户使用 Simulink 子系统建立系统模型时，有几个简单的操作比较常用，这里做一个简单的列举：

(1) 子系统命名：命名方法与模块命名类似。为增强系统模型的可读性，应使用有代表意义的文字给子系统进行命名。

(2) 子系统编辑：用鼠标左键双击子系统模块图标，打开子系统以对其进行编辑。

(3) 子系统的输入：使用 Sources 模块库中的 Inport 输入模块（即 In1 模块）作为子系统的输入端口。

(4) 子系统的输出：使用 Sinks 模块库中的 Outport 输出模块（即 Out1 模块）作为子系统的输出端口。

注意：在子系统中，输入端口 In1 模块与输出端口 Out1 模块属于虚模块，它们的作用在于传递与标记信号（如信号标签的传递、标记子系统的输入输出等），而对信号本身没有任何改变。

7.2 Simulink 高级子系统技术

子系统最基本的应用是将一组相关的模块封装到一个单一的模块之中，以利于用户建立和分析系统模型。在前面的介绍之中，无论是使用 Subsystems 模块库中的 Subsystem 模块，还是直接对已有的模块生成子系统，子系统都可以看作是具有一定输入输出的单个模块，其输出直接依赖于输入的信号；也就是说，对于一定的输入，子系统必定会产生一定的输出。但是在有些情况下，只有满足一定的条件时子系统才被执行；也就是说子系统的执行依赖于其它的信号，这个信号称之为控制信号，它从子系统单独的端口即控制端口输入。这样的子系统称之为条件执行子系统。在条件执行子系统中，子系统的输出不仅依赖于子系统本身的输入信号，而且还受到子系统控制信号的控制。

条件执行子系统的执行受到控制信号的控制，根据控制信号对条件子系统执行的控制方式的不同，可以将条件执行子系统划分为如下的几种基本类型。

(1) 使能子系统：是指当控制信号的值为正时，子系统开始执行。

(2) 触发子系统：是指当控制信号的符号发生改变时（也就是控制信号发生过零时），子系统开始执行。触发子系统的触发执行有三种形式：

① 控制信号上升沿触发：控制信号具有上升沿形式。

② 控制信号下降沿触发：控制信号具有下降沿形式。

③ 控制信号的双边沿触发：控制信号在上升沿或下降沿时触发子系统。

(3) 函数调用子系统：这时条件子系统是在用户自定义的 S-函数中发出函数调用时开始执行。有关 S-函数的概念将在后续章节中介绍。

下面对不同类型的条件执行子系统进行详细的介绍。

7.2.1　条件执行子系统的建立方法

在进一步介绍条件执行子系统之前，首先介绍如何建立条件执行子系统（如图 7.3 所示）。其中需要使用 Subsystems 模块库中的 Enabled Subsystem（使能子系统）模块、Triggered Subsystem（触发子系统）模块及 Enabled and Triggered Subsystem（使能触发子系统）模块。

在建立条件执行子系统前需要注意以下两点：

(1) 对于 Simulink 的早期版本而言，不存在专门的 Subsystems 模块库。如果需要建立条件执行子系统，用户可以在给定的子系统中加入 Signals and Systems 模块库中的 Enable 使能信号与 Trigger 触发信号，以使子系统成为使能子系统、触发子系统或使能触发子系统。

(2) Simulink 系统模型的最高层不允许使用 Enable 与 Trigger 信号，而仅允许在子系统中使用。所以在 Simulink 的后期版本中，Signals and Systems 模块中不再包含 Enable 与 Trigger 信号，而只有在 Subsystems 模块库中的 Enabled Subsystem 模块、Triggered Subsystem 模块以及 Enabled and Triggered Subsystem 模块中才含有此二信号。

图 7.3 中并没有建立一个完整的动态系统的模型，而仅仅是给出建立条件执行子系统的方法，因此并没有给出执行系统所需的使能信号源与触发信号源（如图 7.3 中椭圆曲线所示，

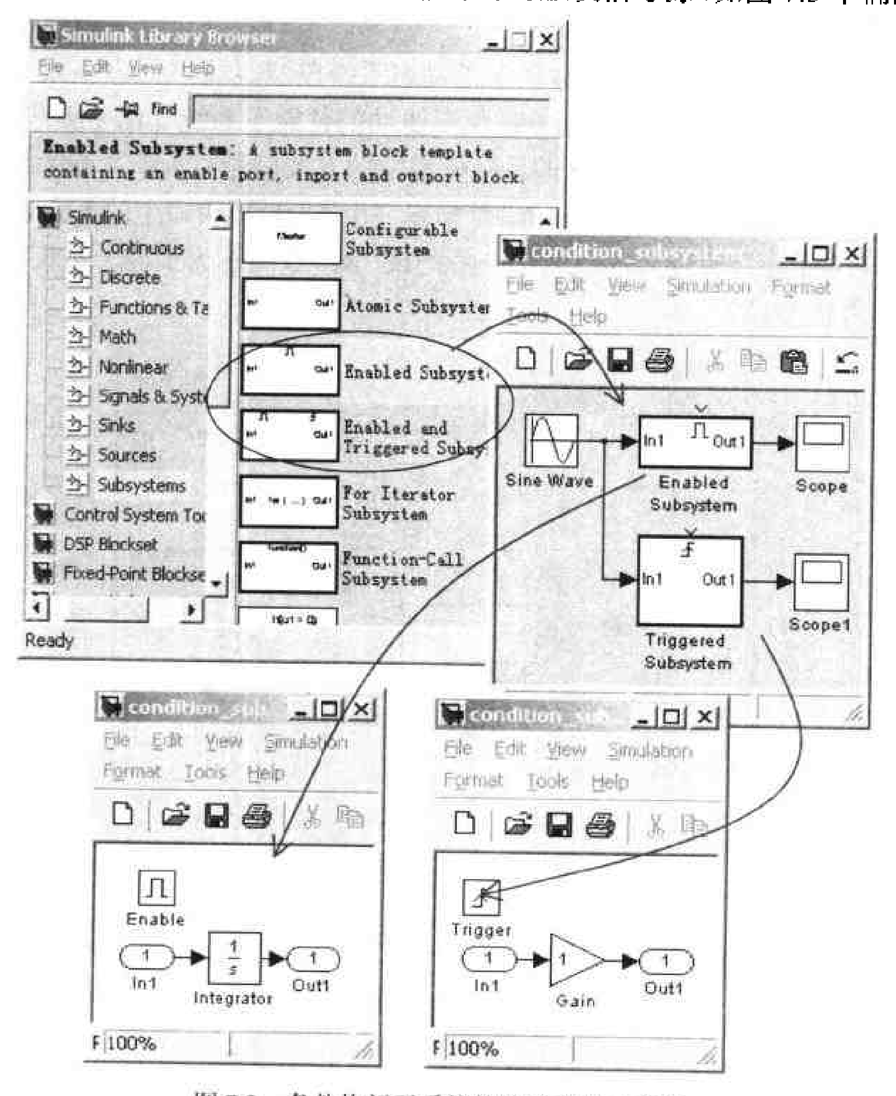


图 7.3　条件执行子系统的建立方法示意图

使能输入端与触发输入端采用不同的信号标志）。如果此时用户运行此系统进行仿真，MATLAB 命令窗口中会给出输入端口没有信号连接的警告，而且系统的输出均为 0。

对于实际的动态系统而言，其中某些子系统很可能受到多个控制信号的控制（也就是系统输出受到多个条件的约束），因此在相应的系统模型建立时，应该使用多个控制信号输入。用户在建立这样的条件执行子系统时，需要注意的是，在一个系统模块中不允许有多个 Enable 信号或 Trigger 信号。如果需要多个控制信号相组合，用户可以使用逻辑操作符来实现。

条件执行子系统是一个应用非常广泛的子系统。其实在 Simulink 的 Subsystems 子系统模块库中还存在着大量的其它类型的子系统。这些子系统均可以看作是某种形式的条件执行子系统，但是对它的控制并非使用使能信号或触发信号就能实现。

7.2.2 使能子系统

在对具体的条件执行子系统进行介绍时，采用实际的系统模型进行说明非常有利于读者的理解。因此，本书均以实际的系统模型为例进行说明。

所谓的使能子系统，是指只有当子系统的使能信号输入为正时，子系统才开始执行。

【例 7.1】 使能子系统的建立与仿真。

按照 7.1 节中的方法建立如图 7.4 所示的动态系统模型。

在此系统模型中，存在着两个由方波信号驱动的使能子系统（图中虚线框所框的子系统，以 A 与 B 表示）。当控制信号（即系统模型中的方波信号）为正时开始执行子系统 A，控制信号为负时（方波信号经过一反相信号操作，由 Math 模块库中的 Logical Operator 逻辑操作模块 NOT 操作符实现）开始执行子系统 B。图 7.5 所示为使能子系统 A 与 B 的结构以及相应的使能状态设置。

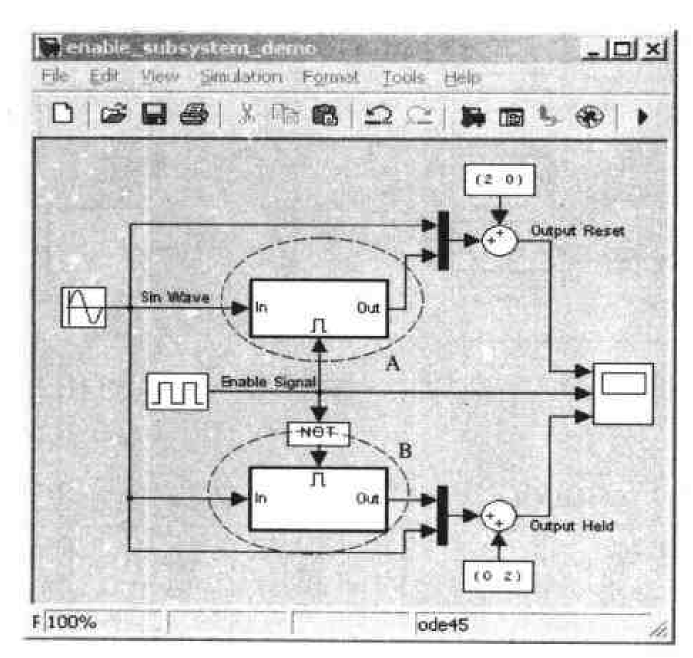


图 7.4 使能子系统模型

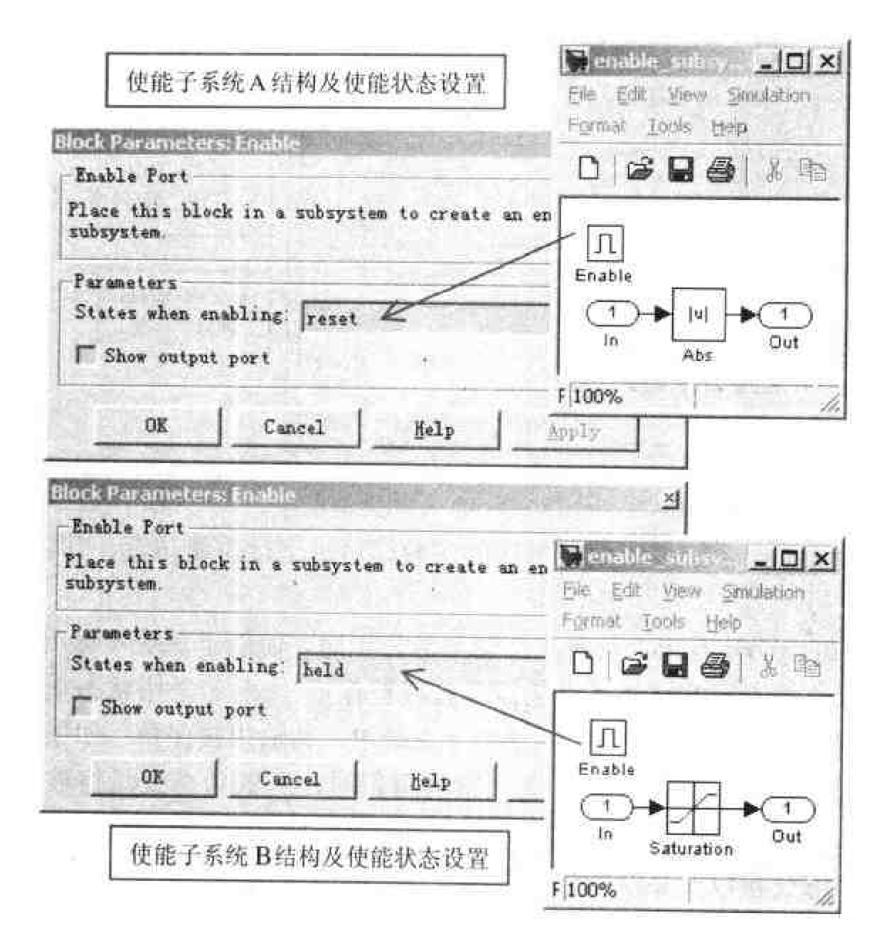


图 7.5　使能子系统 A、B 的内部实现及使能参数设置

在使能信号的使能状态设置中，选择状态重置 reset 表示在使能子系统开始执行时，系统中的状态被重新设置为初始参数值；而状态保持 held 表示在使能子系统开始执行时，系统中的状态保持不变。

至于整个系统的输入输出以及各使能子系统本身的功能，对于用户来说是非常简单的，这里不再赘述。下面给出系统中各个模块的参数设置及仿真参数设置，然后进行仿真分析。

此系统模型中各模块的参数设置如下：

(1) 系统输入为采用默认设置的正弦信号（即单位幅值，单位频率的单位正弦信号）。

(2) 使能子系统的控制信号源，使用 Sources 模块库中的 Pulse Generator 脉冲信号发生器所产生的方波信号。其设置为：脉冲周期（Period）为 5 s，其余采用默认设置。

(3) 使能子系统 A 中的使能信号，其使能状态设置为重置 reset；使能子系统 B 中的使能信号，其使能状态设置为保持 held。

(4) 下方使能子系统中饱和模块（Saturation），其参数设置为：饱和上限为 0.75，饱和下限为-0.75。

(5) 偏移常数信号，其参数设置分别为[2　0]与[0　2]，如图 7.4 中系统模型所示。

(6) 系统输出 Scope 模块参数设置，如图 7.6 所示。

系统仿真参数设置如下：

(1) 仿真时间：设置仿真时间范围为 0 至 20 s。

(2) 求解器设置：采用默认设置，即连续变步长，具有过零检测能力的求解器。

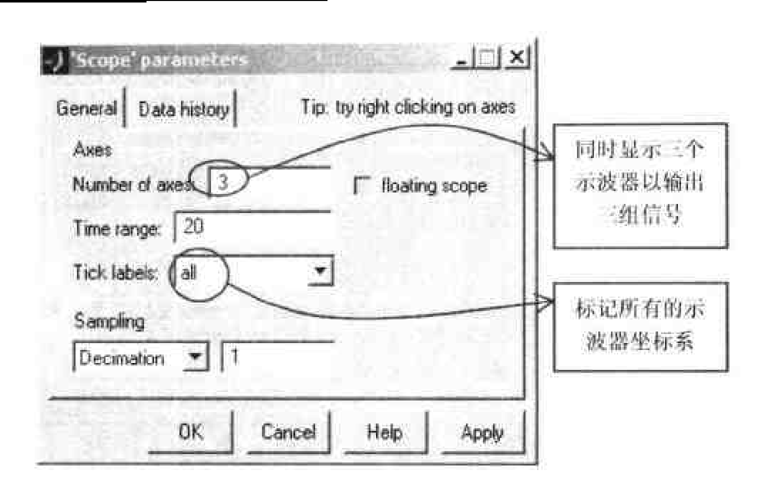


图 7.6　Scope 模块参数设置

其它参数设置采用系统模块中所默认的参数即可。运行系统仿真，其仿真结果如图 7.7 所示。

从图 7.7 中可以明显看出，只有在控制信号为正时，使能子系统才输出，而且设置不同的使能状态可以获得不同的结果（结果被重置或被保持）。对于采用状态重置的使能子系统 A，其输出被重置；而采用状态保持的使能子系统 B，其输出被保持。如果在使能子系统中存在着状态变量，那么当使能模块状态设置为重置时，它的状态变量将被重置为初始状态（可能不为零），当使能模块设置为保持时，它的状态变量将保持不变。至于系统的输出，取决于系统的状态变量以及系统的输入信号，这里不再赘述。

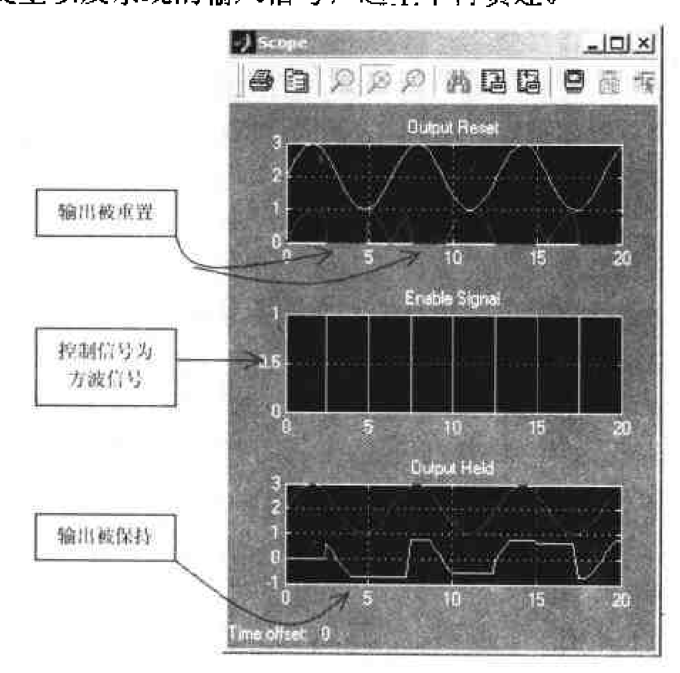


图 7.7　【例 7.1】中使能子系统的仿真结果

7.2.3　触发子系统

所谓的触发子系统指的是只有在控制信号的符号发生改变的情况下（也就是控制信号出现过零事件时），子系统才开始执行。如前所述，根据控制信号符号改变方式的不同可以

将触发子系统分为如下三类：

(1) 上升沿触发子系统。系统在控制信号出现上升沿时开始执行。

(2) 下降沿触发子系统。系统在控制信号出现下降沿时开始执行。

(3) 双边沿（上升沿或下降沿）触发子系统。系统在控制信号出现任何过零时开始执行。

下面以具体的系统实例来说明触发子系统的特点。

【例 7.2】　触发子系统的建立与仿真分析。

在这个例子中，存在着三个使用不同触发方式的触发子系统，分别是上升沿触发、下降沿触发以及双边沿触发。图 7.8 所示为此系统的系统模型。

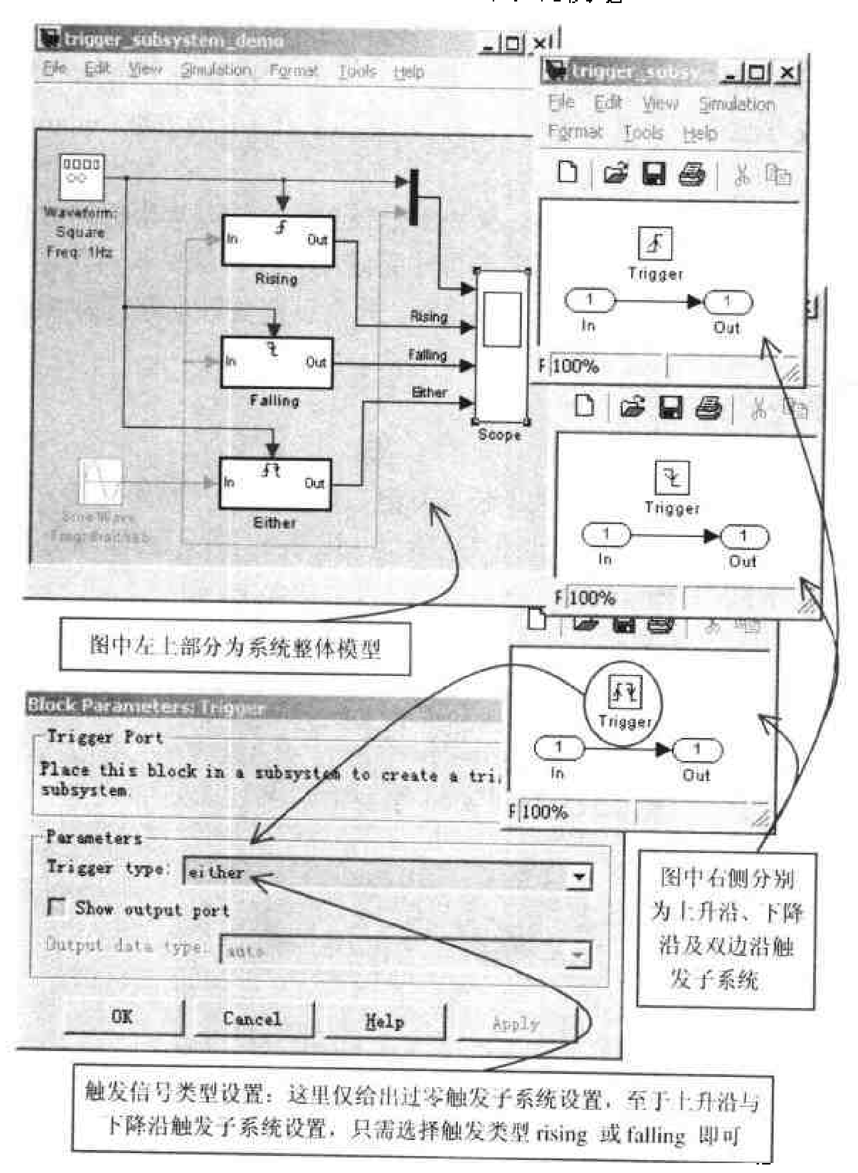


图 7.8　触发子系统的系统模型与触发类型设置

为了突出触发控制信号的作用以及触发子系统的特点，这个例子中的三种不同类型的触发子系统其输入与输出之间直接相连，而没有其它的任何模块。需要注意的是，对于触发子系统而言，它们都具有零阶保持的特性。所谓的零阶保持，是指输出结果保持不变。对于触发子系统而言，系统在触发信号控制下开始执行的时刻，系统由输入产生相应的输出；当触发信号离开过零时，系统的输出保持在原来的输出值，并不发生变化。用户可以

在后面给出的系统仿真结果中明显看出触发子系统的零阶保持特性。此外，由于触发子系统的执行依赖于触发控制信号，因此对于触发子系统而言不能指定常值采样时间（即固定的采样时间），只有带有继承采样时间的模块才能够在触发子系统中应用。在这个系统之中，对于上升沿与下降沿触发子系统来说，其采样周期为两个触发时刻之间的时间间隔（即触发控制方波信号的两个相邻上升沿或下降沿之间的间隔）；对于双边沿触发子系统来说，其采样周期为触发控制方波信号的相邻的上升沿与下降沿之间的间隔。

下面给出系统模型中各个系统模块的参数以及仿真参数设置，最后给出系统仿真结果并进行一定的分析。模块参数设置如下：

(1) 系统输入正弦信号，其模块参数除频率选择为 8 rad/sec 外，其余采用默认的参数。

(2) 系统触发控制信号为方波信号，使用 Sources 模块库中的信号发生器 Signal Generator 模块生成方波信号，其参数设置为：波形 waveform 为方波 square，幅值为 0.5，频率为 1 Hz。

(3) 系统输出 Scope 模块。在 Scope 模块参数设置中，设置其坐标轴数目为 4，以使用 4 个示波器同时显示 4 组信号。其方法与使能子系统中一样，这里不再赘述。

(4) 各触发子系统参数设置如图 7.8 中所示。分别设置其触发方式为 Rising、Falling 及 Either 即可。

系统仿真参数设置如下：

(1) 仿真时间范围 0 至 8 s。

(2) 采用变步长连续求解器，最大步长为 0.01，以避免信号的不连续。

运行此系统进行仿真，仿真结果如图 7.9 所示。其中第一个示波器输出为系统输入正弦

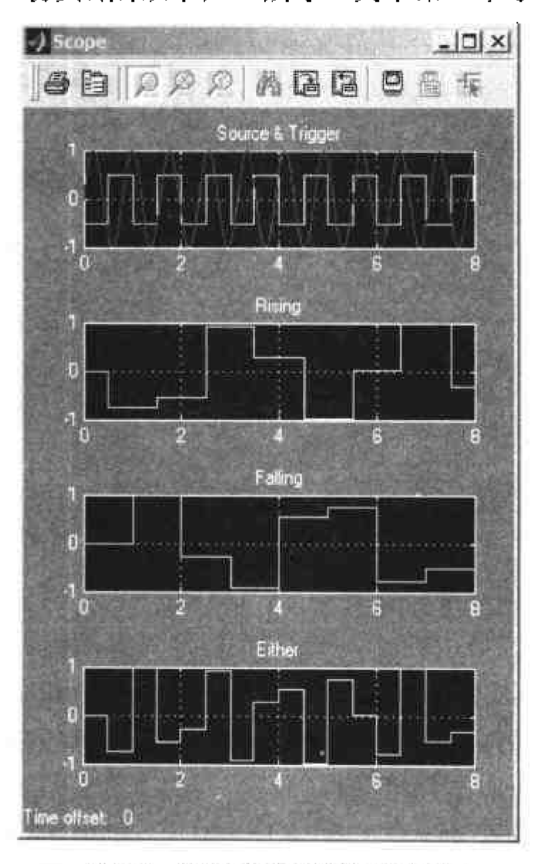


图 7.9 触发子系统的仿真结果

信号以及触发控制方波信号。第二至四个示波器输出分别为上升沿、下降沿与双边沿触发子系统的输出。从仿真结果中可以看出，对于上升沿触发子系统，当方波信号出现上升沿时，系统开始执行，并且其输出在下一个方波信号上升沿到来之前保持不变，即零阶保持特性。对于下降沿触发子系统以及双边沿触发子系统而言，其工作方式与上升沿触发子系统类似，这里不再赘述。

7.2.4 触发使能子系统

在介绍条件执行子系统时已经提到，对于某些条件执行子系统而言，其控制信号可能不止一个。在很多情况下，条件执行子系统同时具有触发控制信号与使能控制信号，这样的条件执行子系统一般称之为触发使能子系统。顾名思义，触发使能子系统指的是子系统的执行受到触发信号与使能信号的共同控制，也就是说，只有当触发条件与使能条件均满足的情况下，子系统才开始执行。触发使能子系统的工作原理如图 7.10 所示。

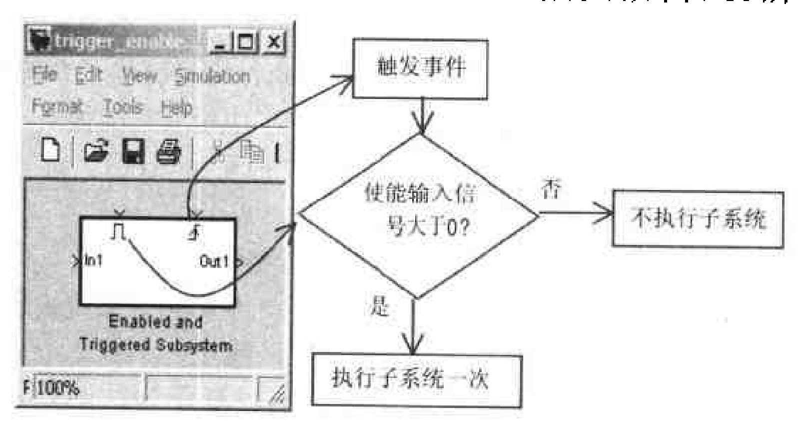


图 7.10　触发使能子系统的工作原理

触发使能子系统的工作原理如下：系统等待一个触发事件的发生（也就是触发信号产生），当触发事件发生后，Simulink 检测使能控制信号，如果使能控制信号为正，则子系统执行一次，否则不执行子系统。由此可知，仅有触发条件与使能条件均满足时，子系统才能够被执行。

对于使用多个控制信号的条件子系统，用户一定要记住：条件执行子系统中不允许同时出现多个 Trigger 信号或 Enable 信号。如果必须使用多个控制信号，用户可以使用逻辑操作符，先将相关的控制信号（即子系统执行条件）相组合，以产生单一的触发控制信号或使能控制信号。

7.2.5 原子子系统

1. 原子子系统的概念

至此，本章已经介绍了通用子系统、使能子系统以及触发子系统等三种不同子系统的概念及其使用。虽然子系统都可以将系统中相关模块组合封装成一个单独的模块，大大方便了用户对复杂系统的建模、仿真及分析；但是对于不同的子系统而言，它们除了有如上的共同点之外，还存在着本质上的不同。下面介绍此三种子系统的不同本质，并简单介绍原子子系统的概念。

众所周知，无论是对通用子系统还是使能子系统，Simulink 在进行系统仿真时，子系统中的各个模块的执行并不受到子系统的限制。也就是说，系统的执行与通用子系统或使能子系统的存在与否无关。这两种子系统的使用均是为了使 Simulink 框图模型形成一种层次结构，以增强系统模型的可读性。子系统中的模块在执行过程中与上一级的系统模块统一被排序，模块的执行顺序与子系统本身无关，在一个仿真时间步长之内，系统的执行可以多次进出同一个子系统。这与在前面所介绍的用于信号连接（或组合）的“虚块”的概念类似，因此，对于通用子系统与使能子系统这两种子系统，我们称之为“虚子系统”。子系统相当于一种虚设的模块组容器，其中的各模块与系统中其它模块（子系统）的信号输入输出不受到任何的影响。简单来说，虚子系统具有如下的特点：

(1) 子系统只是系统模型中某些模块组的图形表示。

(2) 子系统中的模块在执行时与其上一级模块统一被排序，不受子系统的限制。

(3) 在一个仿真时间步长内，Simulink 可以多次进出一个子系统。

对于触发子系统而言，其工作原理与上述两种子系统的概念不同。在触发子系统中，当触发事件发生时，触发子系统中的所有模块一同被执行。只有当子系统中的所有模块都被执行完毕后，Simulink 才会转移到系统模型中的上一层执行其它的模块。这与上述子系统中各模块的执行方式根本不同。这样的子系统称之为“原子子系统”。原子子系统具有如下的特点：

(1) 子系统为一“实际”的模块，需要按照顺序连续执行。

(2) 子系统作为一个整体进行仿真，其功能类似于一个单独的系统模块。

(3) 子系统中的模块在子系统中被排序执行。

2. 建立原子子系统

触发子系统在触发事件发生时，子系统中的所有模块一同被执行，当所有的模块都执行完毕后，Simulink 才开始执行上一级其它模块或其它子系统，因此触发子系统为一原子子系统。但是在有些情况下，需要将一个普通的子系统作为一个整体进行执行，而不管它是不是触发子系统。这对于多速率复杂系统尤其重要，因为在多速率系统中，时序关系的任何差错，都会导致整个系统设计仿真的失败，而且难以进行诊断分析，尤其是在要生成系统可执行代码时更为重要。

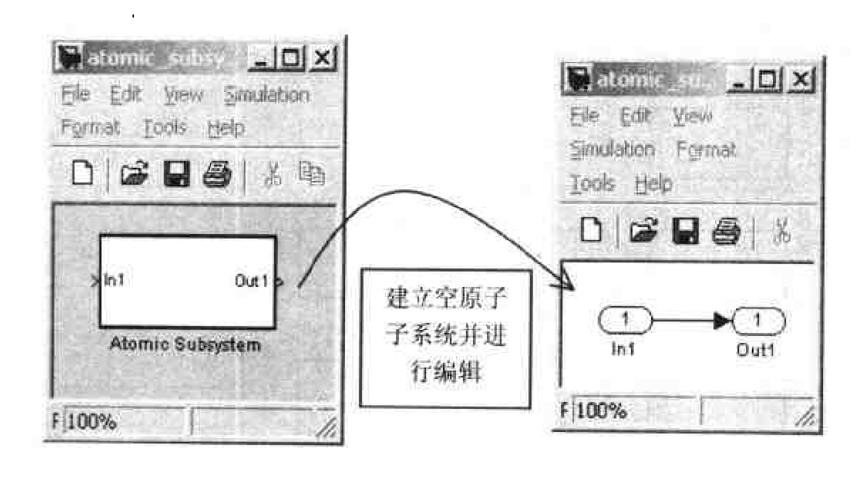
在 Simulink 中有两种方法可以建立原子子系统：

(1) 建立一个空的原子子系统。选择使用 Subsystems 模块库中的 Atomic Subsystem 子系统模块，然后编辑原子子系统。

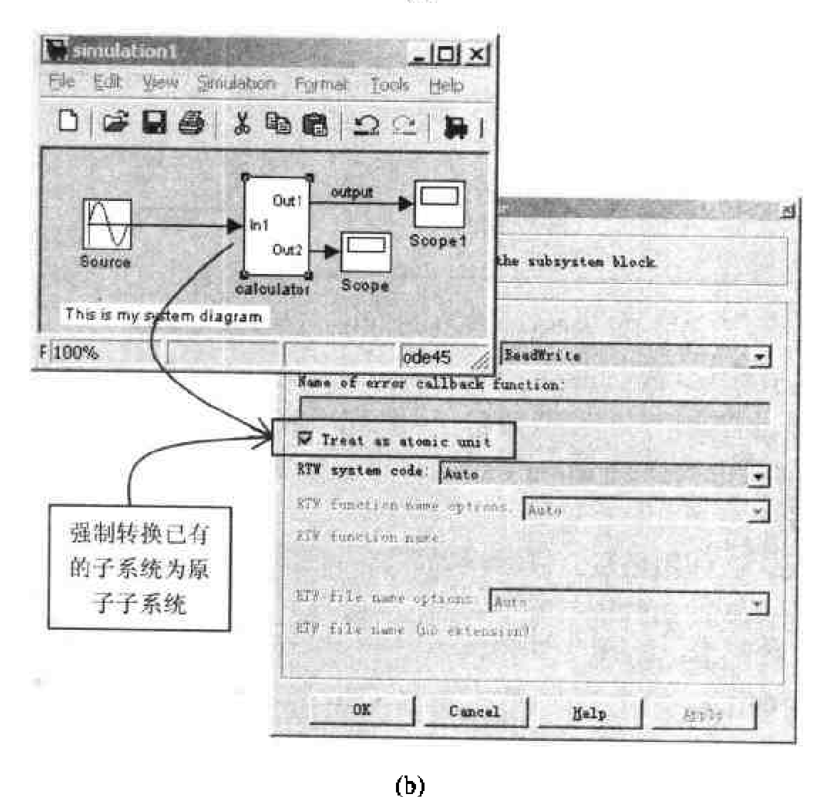
(2) 将已经建立好的子系统强制转换为原子子系统。首先选择子系统，然后选择 Simulink 模型编辑器的 Edit 菜单下的 Block Parameters 模块参数，框选 Treat as Atomic Unit（作为原子子系统）即可，或单击鼠标右键，在弹出菜单中选择相同选项也可。

原子子系统的建立方法如图 7.11(a)、7.11(b)所示。

注意：对于使能子系统而言，不能将其转换为原子子系统，这是因为使能子系统中模块执行顺序不能被改变。



(a)



(b)

图 7.11　原子子系统的建立方法

(a) 使用 Atomic Subsystem 模块建立原子子系统；(b) 设置已有子系统为原子子系统

7.2.6　其它子系统介绍

在 Simulink Block Library（Simulink 模块库，版本 4.1）中的 Subsystems 子系统模块库中除了前面所介绍的通用子系统、触发子系统、使能子系统以及原子子系统之外，Simulink 还提供了许多其它的条件执行子系统。图 7.12 所示为 Subsystems 模块库中的所有子系统模块。在此对其进行简单的介绍。

(1) 可配置子系统（Configurable Subsystem）：用来代表用户自定义库中的任意模块，只能在用户自定义库中使用。

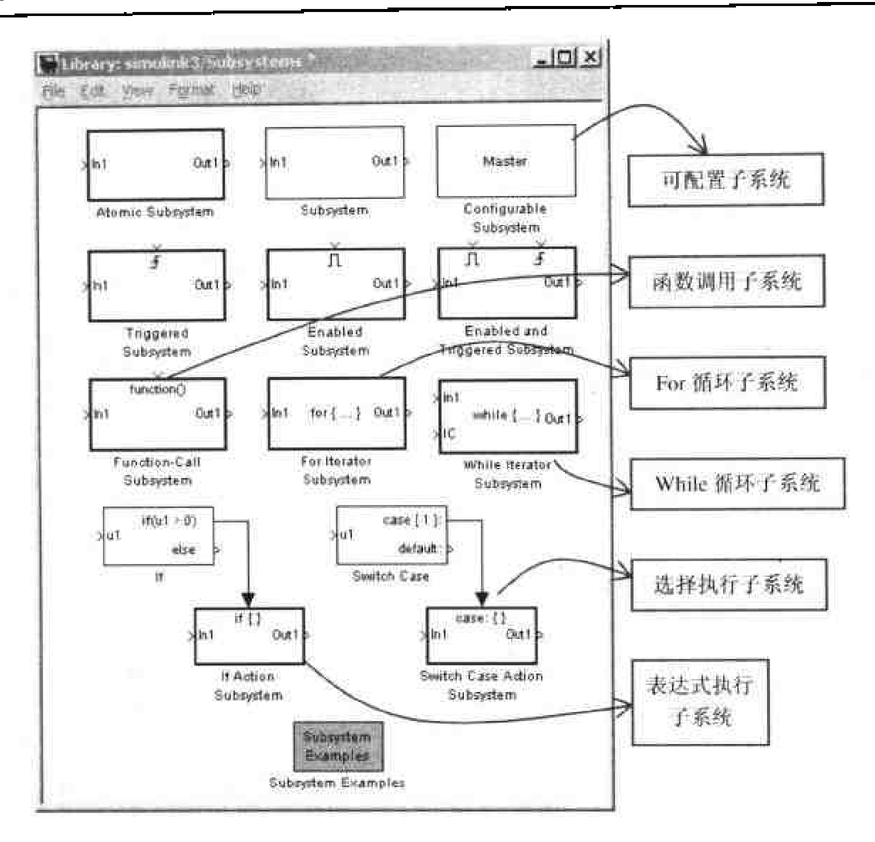


图 7.12　Subsystems 模块库中的所有子系统模块

(2) 函数调用子系统（Function-Call Subsystem）：使用 S-函数的逻辑状态而非普通的信号作为触发子系统的控制信号。函数调用子系统属于触发子系统，在触发子系统中触发模块 Trigger 的参数设置中选择 Function-Call 可以将由普通信号触发的触发子系统转换为函数调用子系统，如图 7.13 所示。

需要注意的是，在使用函数调用子系统时，子系统的函数触发端口必须使用 Signals & Systems 模块库中的函数调用发生器 Function-Call Generator 作为输入。这里需要使用 S-函数（至于 S-函数的生成与编写，将在第 9 章中介绍）。

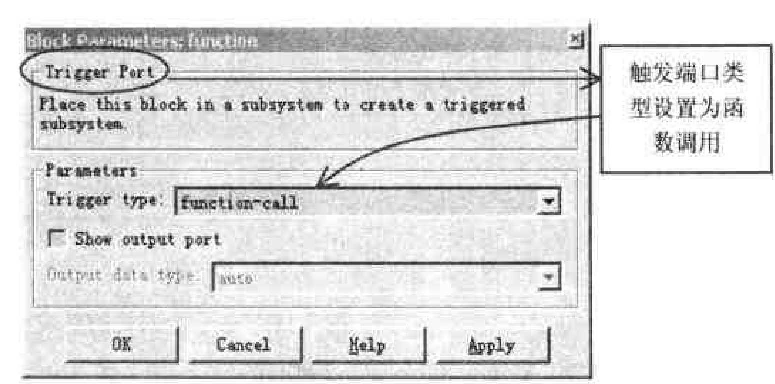


图 7.13　函数调用触发类型设置

(3) For 循环子系统（For Iterator Subsystem）：For 循环子系统的目的是在一个仿真时间步长之内循环执行子系统。用户可以指定在一个仿真时间步长内子系统执行的次数，以达到某种特殊的目的。有兴趣的读者可以参考 Simulink 的示例：sl_subsys_for1.mdl，在 MATLAB 命令窗口中键入文件名即可打开此系统模型。

(4) While 循环子系统（While Iterator Subsystem）：与 For 循环子系统相类似，While 循环子系统同样可以在一个仿真时间步长内循环执行子系统，但是其执行必须满足一定的条件。While 循环子系统有两种类型：当型与直到型，这与其它高级语言中的 While 循环类似。

(5) 选择执行子系统（Switch Case Action Subsystem）：在某些情况下，系统对于输入的不同取值，分别执行不同的功能。所谓的选择执行子系统，与 C 语言中的 Switch Case 语句的功能类似。需要注意的是，选择执行子系统必须同时使用 Switch Case 模块与 Switch Case Action Subsystem 模块（均在 Subsystems 模块库中）。

(6) 表达式执行子系统（If Action Subsystem）：为了与前面的条件执行子系统相区别，这里我们称 If Action Subsystem 为表达式执行子系统。此子系统的执行依赖于逻辑表达式的取值，这与 C 语言中的 If Else 语句类似。需要注意的是，表达式执行子系统必须同时使用 If 模块与 If Action Subsystem 模块（均在 Subsystems 模块库中）。

这里仅仅对 Simulink 中 Subsystems 子系统模块库中的其它子系统模块的功能进行了简单的说明，并未涉及如何使用。如果读者对此感兴趣，可以参考 Subsystems 模块中的 Subsystem Examples 子系统实例。限于篇幅，这里不再赘述。

7.3 Simulink 的子系统封装技术

第 5 章中介绍了如何将模块组构成一个子系统。使用子系统技术可以很好地改善系统模型的界面。然而在使用 Simulink 进行系统仿真分析时，首先需要进行模块参数设置，因此需要对子系统中所有的模块进行正确的参数设置。在前面的介绍中，我们均是逐一设置子系统中各模块的参数，这不可避免地给用户带来了很多的不便。因为子系统一般均为具有一定功能的模块组的集合，在系统中相当于一个单独的模块，具有特定的输入与输出关系。对于已经设计好的子系统而言，如果能够像 Simulink 模块库中的模块一样进行参数设置，则会给用户带来很大的方便，这时用户只需要对子系统参数选项中的参数进行设置，而无需关心子系统的内部模块的实现。Simulink 的子系统封装技术可以实现这一点，从而大大方便了用户的使用。

7.3.1 如何封装子系统

封装子系统与建立子系统并不相同，建立子系统指的是将具有一定功能的一组模块“容纳”在一个子系统之中，使用单一图形方式的子系统模块来表示一组模块，从而增强系统模型的可读性，在动态系统进行仿真时需要对子系统中各个模块的参数分别进行设置；而封装子系统指的是将已经建立好的具有一定功能的子系统进行封装，封装的目的在于生成用户自定义的模块，此模块与子系统的功能完全一致。通过定义用户自己的图标、参数设置对话框以及帮助文档等等，可以使封装后的子系统与 Simulink 中内置的系统模块具有相同的操作：双击封装后的子系统模块以打开模块参数设置对话框进行参数设置，将系统仿真所需要的参数传递到子系统之中；同时可以查看模块的帮助文档以获得子系统输入输出关系、子系统功能以及模块描述等帮助信息。除此之外，封装后的子系统还拥有自己的工作区。这样使得用户建立的系统模型框图更为专业，并且可以保护子系统的内部实现，从而使得对子系统的操作与通用 Simulink 模块一样友好。

简单来说，封装子系统具有如下特点：

(1) 自定义子系统模块及其图标。

(2) 用户双击封装后的图标时显示子系统参数设置对话框。

(3) 用户自定义子系统模块的帮助文档。

(4) 封装后的子系统模块拥有自己的工作区。

因此，使用封装子系统技术具有以下优点：

(1) 向子系统模块中传递参数，屏蔽用户不需要看到的细节。

(2) “隐藏”子系统模块中不需要过多展现的内容。

(3) 保护子系统模块中的内容，防止模块实现被随意篡改。

下面我们以实际的例子来说明如何进行子系统的封装，并全面介绍子系统的封装技术。

【例 7.3】 以第 5 章中人口动态变化的非线性离散模型为例说明子系统的封装技术。

解：封装子系统的基本过程如下：

(1) 打开人口动态变化的非线性离散模型框图。

(2) 生成需要进行封装的子系统。

(3) 选择需要封装的子系统，单击鼠标右键选择 Mask subsystem，或使用 Edit 菜单项中的相应命令进行子系统封装。

封装子系统的基本流程图如图 7.14 所示，图中上方为系统原始模型框图，中间为使用

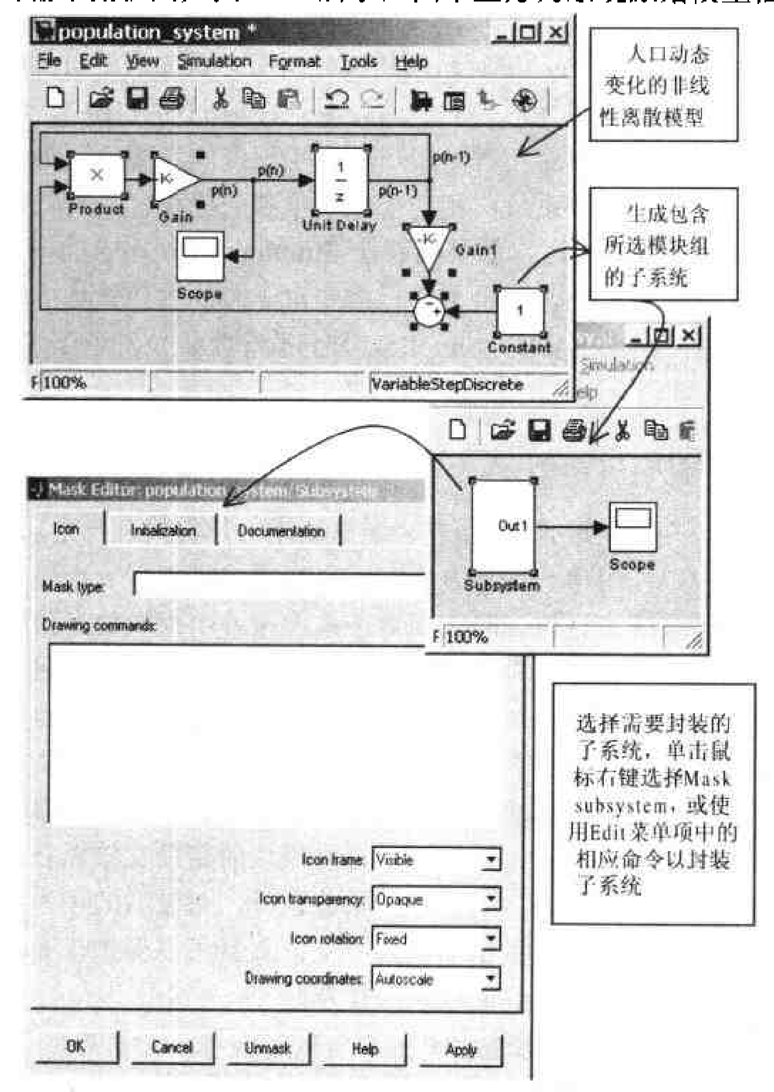


图 7.14 子系统封装流程示意图

子系统的系统模型框图。当选择 Mask subsystem 菜单命令后将出现图中下方所示的封装编辑器。使用封装编辑器可以编辑封装后子系统模块的图标（Icon）、参数初始化设置对话框（Initialization）以及帮助文档（Documentation），从而可以使用户设计出非常友好的模块界面，以充分发挥 Simulink 的强大功能。

7.3.2 封装编辑器之图标编辑对话框

当选择 Mask subsystem 菜单命令进行子系统封装时，将出现如图 7.15 所示的封装编辑器并显示图标编辑对话框。

使用此编辑器可以对封装后的子系统进行各种编辑。这里首先介绍图标编辑对话框的功能与使用。

在默认情况下，封装子系统不使用图标。但友好的子系统图标可使子系统的功能一目了然。为了增强封装子系统的界面友好性，用户可以自定义子系统模块的图标。只需在图标编辑对话框中的子系统模块图标绘制命令栏（Drawing Commands）中使用 MATLAB 中相应的命令便可绘制模块图标，并可设置不同的参数控制图标界面的显示。下面逐一介绍各对话框的使用。

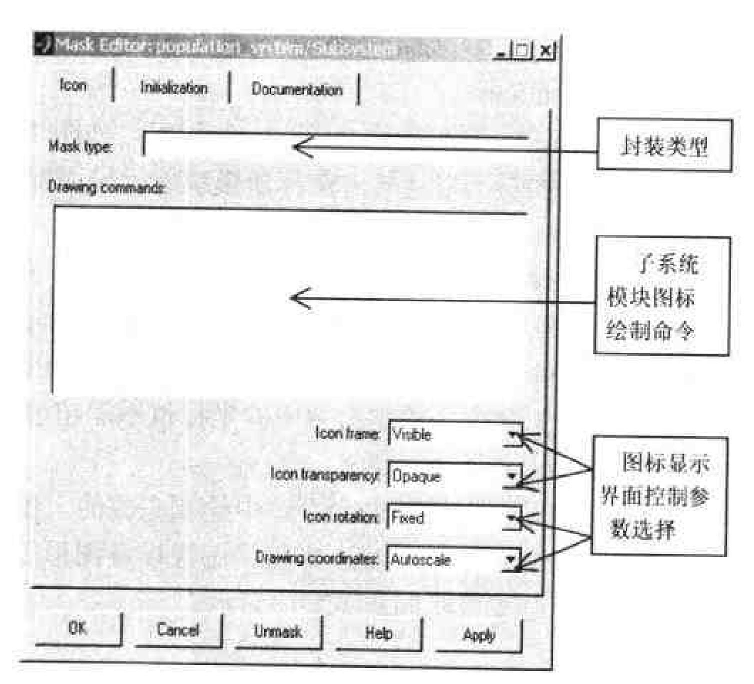


图 7.15　封装编辑器之图标编辑对话框

1. 封装类型（Mask Type）

封装类型用来对封装后的子系统进行简短的说明。在此例中，用户可以键入 Population Sample Mask。它将显示在参数对话框的左上角。

2. 图标显示界面控制参数

通过设置不同的参数可使模块图标具有不同的显示形式。控制参数共有四种。

1）图标边框设置（Icon frame）

功能：设置图标边框为可见（Visible）或不可见（Invisible），如图 7.16 所示，其中左侧表示图标边框可见，而右侧表示边框不可见。

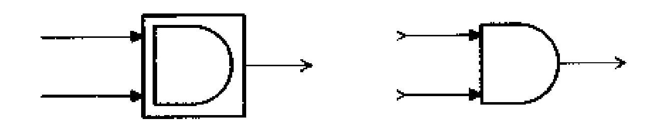


图 7.16　边框设置示意图

2) 图标透明性设置（Icon transparency）

功能：设置图标为透明（Transparency）或不透明（Opaque）显示，如图 7.17 所示，其中左侧为图标不透明，而右侧表示图标透明（此时在图标后面的内容如模块端口标签可以被显示出）。

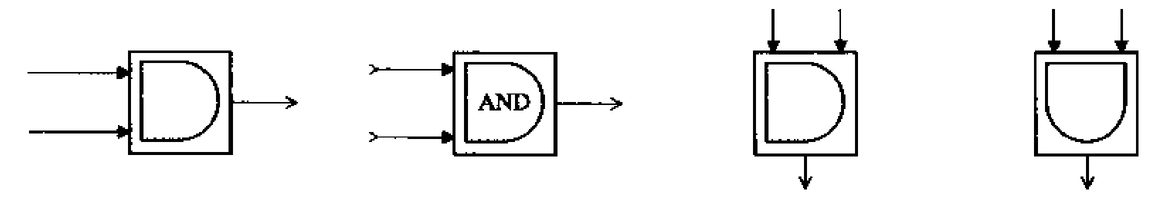


图 7.17　图标透明显示设置示意　　图 7.18　图标旋转显示设置示意图

3) 图标旋转性设置（Icon rotation）

功能：设置图标为固定（Fixed）或可旋转（Rotates）显示，如图 7.18 所示，其中左侧表示图标不可旋转，而右侧表示图标可以旋转（图标随模块的旋转而旋转）。

4) 图标绘制坐标系设置（Drawing coordinates）

功能：设置图标绘制命令所使用的坐标系单位，仅对 plot 与 text 命令有效。其选项分别为自动缩放（Autoscale）、像素（Pixels）以及归一化表示（Normalized）。其中 Autoscale 表示图标自动适合模块大小，与其成比例缩放；Pixels 表示图标绘制采用像素作为单位；Normalized 表示模块的大小为单位长度，绘制命令中的坐标值不能超过单位值 1。

3. 图标绘制命令栏（Drawing commands）

封装后子系统模块的图标均是在图标绘制命令栏中绘制完成的。使用不同的绘制命令可以生成不同的图标如描述性的文本、系统状态方程、图像以及图形等。如果在此栏中键入多个绘制命令，则图标的显示会按照绘制命令顺序显示。

1) 图标为描述性文本

使用如下的绘制命令可以在模块图标上显示文本：

```
disp('text')                         % text 表示图标文本
disp(variablename)                   % variablename 为工作空间中的字符串变量名
text(x, y, 'text')
text(x, y, stringvariablename)       % stringvariablename 为已存在的字符串变量名
text(x, y, text, 'horizontalAlignment', halign, 'verticalAlignment', valign)
%halign 与 valign 分别表示文本水平与垂直对齐方式，其取值不再赘述
fprintf('text')
fprintf('format', variablename)      % format 表示文本的格式
```

port_label(port_type, port_number, label)

% 此命令可以显示模块的端口名称，其中 port_type 为端口类型，取值为'input'或'output'

% port_number 为端口数目，label 为端口文本

如果需要显示多行文本，可以使用\n 表示换行。这时封装后的子系统图标为描述性文本，如图 7.19 所示。

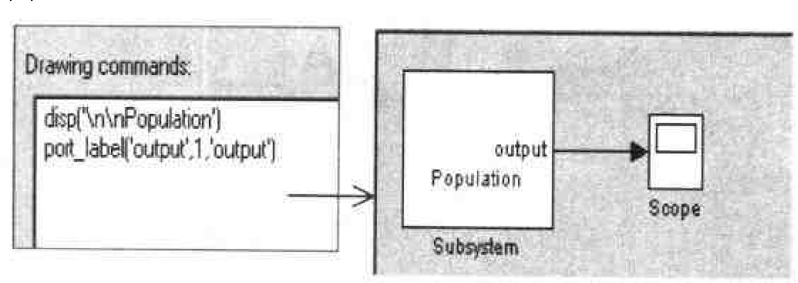


图 7.19　图标绘制为文本

2) 图标为系统状态方程

使用 dpoly 命令在设置封装后子系统模块的图标为系统传递函数，或采用 droots 命令设置为系统零极点传递函数，其命令格式为

dpoly(num, den)

dpoly(num, den, 'character')

droots(z, p, k)

其中 num、den 分别为分子与分母多项式，'character'（如 s 或是 z）为系统频率变量，z、p、k 分别为零点、极点与系统增益。需要注意的是，num、den、z、p、k 均为 MATLAB 工作空间中已经存在的变量，否则绘制命令的执行将出现错误。绘制封装后子系统的图标为系统传递函数如图 7.20 所示。num=[1]，den=[1 2 1]为 MATLAB 中的变量。

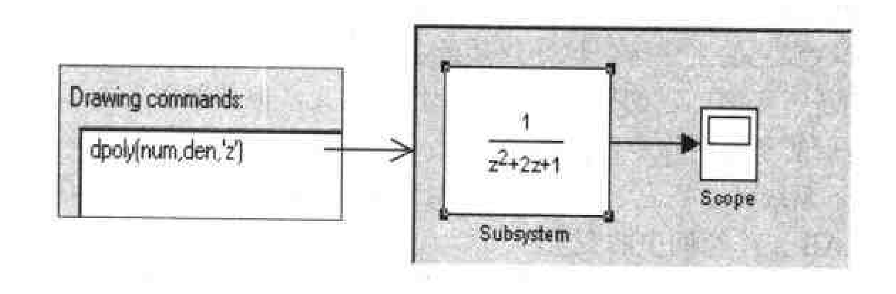


图 7.20　图标绘制为系统传递函数

3) 图标为图像或图形

使用 plot 命令与 image 命令可以设置封装后子系统模块的图标为图形或图像。尽管一般的 MATLAB 命令不能在图标绘制命令栏中直接使用，但它们的返回值可以作为图标绘制命令的参数。绘制封装后子系统模块的图标为图形或图像，如图 7.21 所示。

至于在图标中绘制图像与图形的其它细节，这里不再赘述，感兴趣的读者可以参考 Simulink 的帮助文档。有一点需要注意，在图标中绘制图像时，图像必须为 BMP 格式（否则，需要使用图像处理工具箱中的 ind2rgb 命令进行转换）。此外，所有的绘制命令都可以使用在子系统对话框中输入的参数。

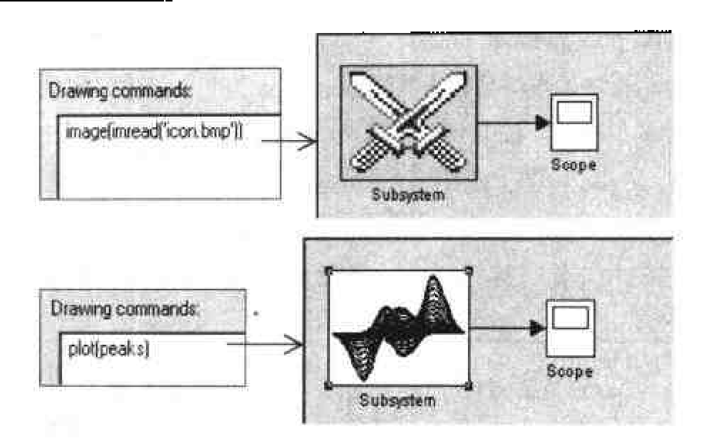


图 7.21　图标绘制为图像与图形

7.3.3　封装编辑器之参数初始化对话框

子系统封装最主要的目的之一便是提供一个友好的参数设置界面。一般的用户无需了解系统内部实现，只需提供正确的模块参数，以使用特定模块的特定功能，从而完成系统设计与仿真分析的任务。如果只是绘制了模块的图标，则模块并没有被真正封装，因为在双击模块时仍显示模块内部的内容，并且始终直接使用来自 MATLAB 工作空间中的参数。只有使用子系统封装编辑器中所提供的参数初始化对话框（Mask Editor 下的 Initialization 选项卡）进行子系统参数输入设置，才可以完成子系统模块的真正封装，从而使用户设计出与 Simulink 内置模块一样直观的参数设置界面。

在没有对子系统进行封装之前，子系统中的模块可以直接使用 MATLAB 工作空间中的变量。通常的子系统均可以被看作是图形化的 MATLAB 脚本，也就是说，子系统只是将一些命令（由模块实现）以图形化的方式组合到一起。而一旦子系统被封装之后，其内部的参数对系统模型中的其它系统不可见，而且只能使用参数设置对话框输入；也就是说，封装后的子系统拥有独立的工作区，这是封装最主要的特点之一。这样用户可以在一个模型中有同样模块的若干实例，它们拥有同样的变量名定义，但其取值各不相同。下面对通常的子系统与封装后的子系统作一个简单的比较：

(1) 通常的子系统可以视为 MATLAB 脚本文件，其特点是子系统没有输入参数，可以直接使用 MATLAB 工作空间中的变量。

(2) 封装后的子系统可以视为 MATLAB 的函数，其特点是封装后的子系统提供参数设置对话框输入参数；不能直接使用 MATLAB 工作空间中的变量；拥有独立的模块工作区（工作空间）；包含的变量对其它子系统及模块不可见；可以在同一模型中使用同样的子系统而其取值可各不相同。

在【例 7.3】中已经介绍了如何将指定的子系统进行封装并进行图标绘制，这里仍以人口变化的非线性离散系统模型为例说明如何对封装后的子系统的参数输入对话框进行设置。在这个人口变化的简单模型中，系统的动力学方程由如下的差分方程来描述：

$$p(n) = rp(n-1)\left[1 - \frac{p(n-1)}{K}\right]$$

其中 $p(n)$ 表示某一年的人口数量，r 表示人口繁殖速率，K 表示新增资源所能满足的个体

数目。因此在封装后子系统模块的参数设置界面中，应该提供相应的初始参数 $p(0)$、繁殖速率 r 以及 K。

使用参数初始化选项卡对子系统模块的参数输入进行设置，如图 7.22 所示。

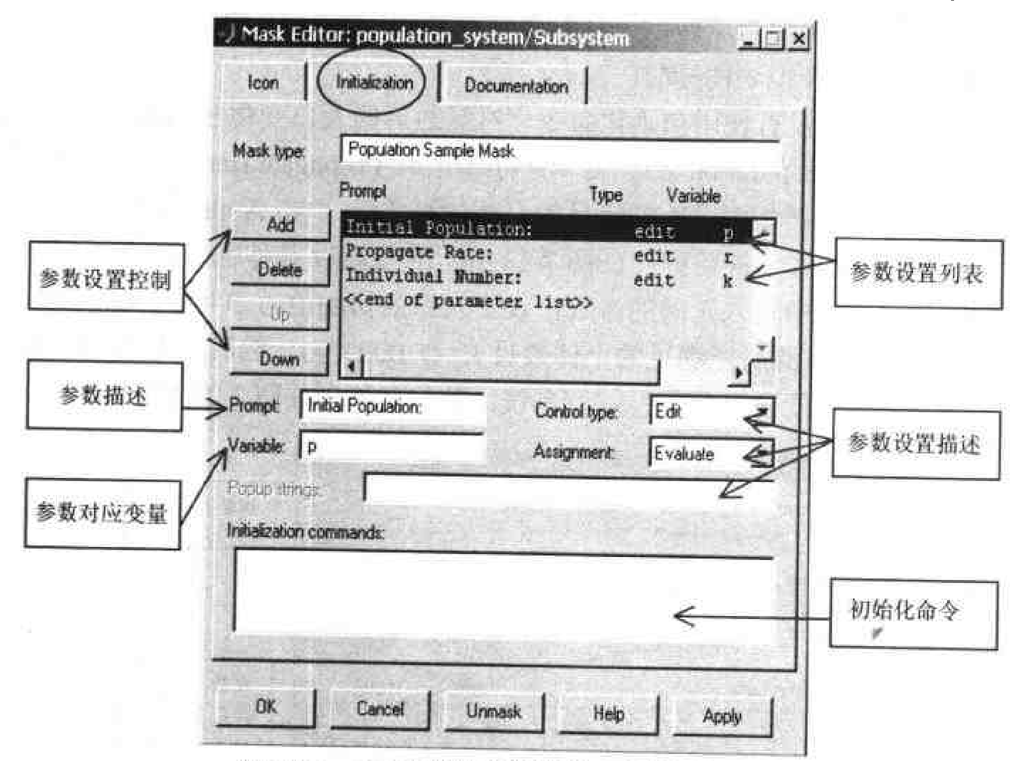


图 7.22　子系统模块参数设置（参数初始化）

下面对参数初始化选项卡的内容作逐一介绍。

1. 参数设置控制

参数设置控制包括添加（Add）、删除（Delete）、上移（Up）与下移（Down），它们分别表示在即将生成的参数设置对话框中添加、删除、上移与下移模块需要的输入参数。

2. 参数描述（Prompt）

参数描述指的是对模块输入的参数作简单的说明，其取值最好能够说明参数的意义或者作用。

3. 参数对应变量（Variable）

参数对应变量表示键入的参数值将传递给封装后的子系统工作空间中相应的变量，在此使用的变量必须与子系统中所使用的变量具有相同的名称。

4. 参数设置描述

参数设置描述包括参数控制类型（Control type）、参数分配类型（Assignment）以及下拉选项框（Popup strings）。其中控制类型包括 Edit（需要用户键入参数值，适合多数情况）、Checkbox（复选框，表示逻辑值）及 Popup（弹出参数选项以供选择取值，弹出参数选项用 Popup strings 栏中由'|'隔开的字符串表示）。参数分配类型表示在 Edit 栏中键入的表达式是作为求值字符串（Evaluate）还是普通的文字字符串（Literal）。

5. 初始化命令栏（Initialization commands）

初始化命令为一般的 MATLAB 命令，在此可以定义封装后子系统工作空间中的各种变

量，这些变量可以被封装子系统模块图标绘制命令、其它初始化命令或子系统中的模块使用。当出现下述情况时，Simulink 开始执行初始化命令：

(1) 模型文件被载入。

(2) 框图被更新或模块被旋转。

(3) 绘制封装子系统模块图标时。

为简单起见，这里没有使用初始化命令。对封装后的人口变化子系统使用图 7.22 中所示的设置（其中 Initial Population 表示人口的初始值 p，Propagate Rate 表示人口繁殖速率 r，Individual Number 表示新增资源所能满足的个体数目 K），单击 Apply 或 OK 按钮，然后双击封装后的子系统模块，这时将弹出如图 7.23 所示的参数设置对话框。

在参数设置对话框中输入正确的参数，设置人口的初始值为 p=100 000、人口繁殖速率为 r=1.05，而新增资源所能满足的个体数目 K=1 000 000。然后单击 Apply 或 OK 按钮并采用与第 5 章中相同的仿真参数。运行系统，其仿真结果如图 7.24 所示。

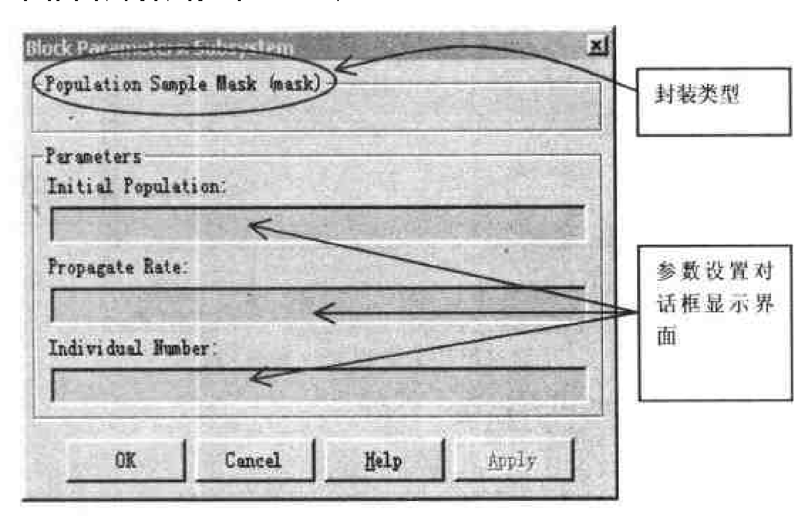


图 7.23 封装后子系统的参数设置对话框

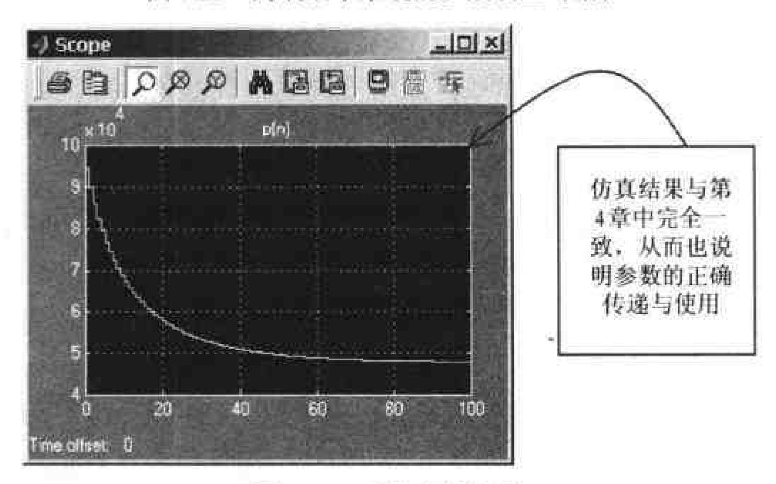


图 7.24 系统仿真结果

7.3.4 封装编辑器之文档编辑对话框

Simulink 模块库中的内置模块均提供了简单的描述与详细的帮助文档，这可以大大方便用户的使用与理解。对于用户自定义的模块（即封装后的子系统），Simulink 提供的文档编

辑功能同样可使用户建立自定义模块的所有帮助文档。图 7.25 所示为封装编辑器中文档编辑选项卡（Documentation），使用文档编辑可以建立用户自定义模块的简单描述文档与模块的详细帮助文档（包括模块的所有信息，可以使用 HTML 格式编写）。当子系统被封装之后，便可以编制子系统模块的描述文档与帮助文档了。对于前面的所封装的人口动态变化的子系统，编写如图 7.25 中所示的文档。

单击 Apply 或 OK 按钮后，双击封装后的模块，则其参数设置对话框中显示模块的描述文档，如图 7.26 所示。

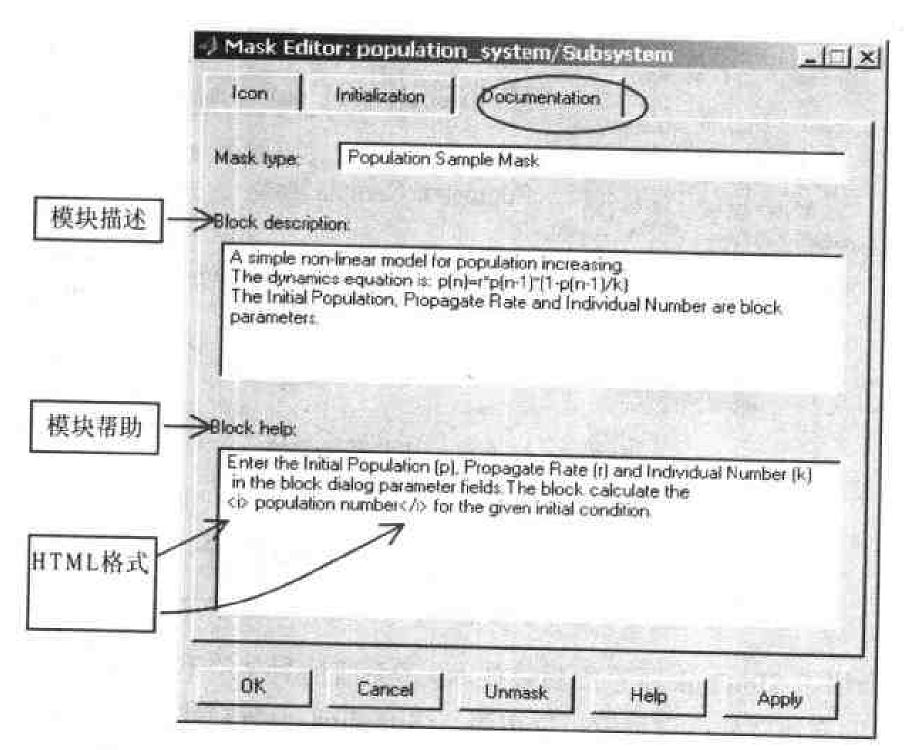


图 7.25　封装编辑器的文档编辑

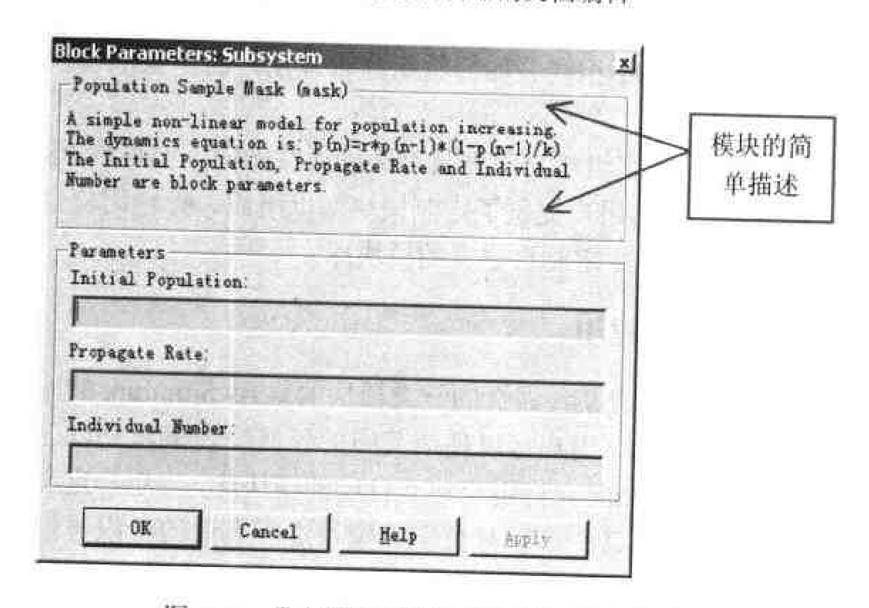


图 7.26　带有模块简单描述的参数设置对话框

编写一个好的文档对于系统的设计与开发往往是至关重要的，它便于用户对系统的使用与维护。如果这时单击 Help 帮助按钮，用户可以 MATLAB 中的帮助系统中获得模块的更进一步的说明与其它的所有相关信息，如图 7.27 所示。

至此，本章已经比较全面地介绍了子系统封装的概念及其使用，下面将进一步介绍使用 Simulink 自定义系统模块库的技术。

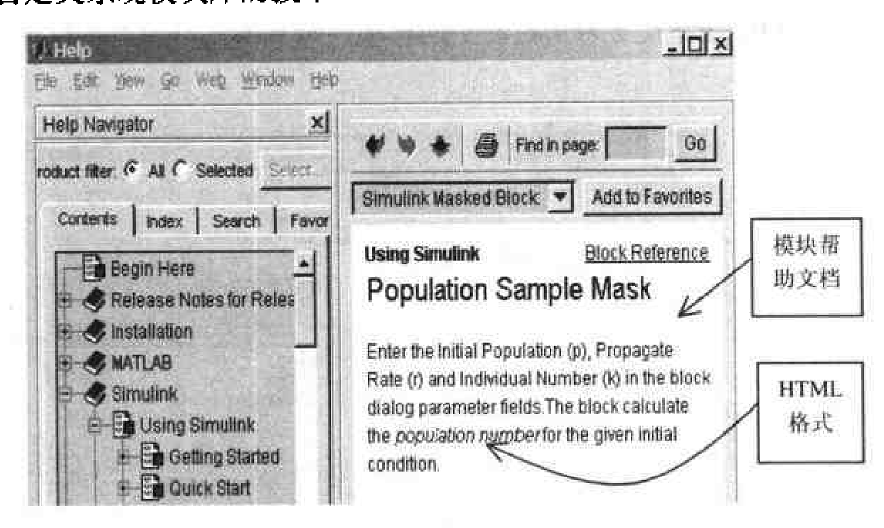


图 7.27 子系统模块的帮助文档

7.4 Simulink 模块库技术

7.3 节中介绍了 Simulink 子系统封装技术。子系统的封装给不同专业领域中的系统设计者带来了很多的方便之处：设计者可以按照与 Simulink 内置模块相同的使用方式使用用户自定义的模块；一旦模块的设计完成并经过正确的测试，用户在使用时便只需要关心模块的输入与输出关系，而无需了解模块内部的实现。然而在很多情况下，用户自定义的模块数量较多，模块的应用范围各不相同。如何组织与管理用户自定义的模块，也是一个比较重要的问题。

Simulink 使用模块库技术来组织与管理具有某种属性的同一类模块，这给用户带来了很大的方便。同时 Simulink 也给用户提供了定制模块库以组织、管理用户自定义的模块。本节将详细介绍如何使用 Simulink 定制自定义的模块库。

7.4.1 模块库的概念及其应用

所谓的模块库一般是指具有某种属性的一类模块的集合。Simulink 的库浏览器中包含了大量的用于不同领域的模块库，用户可以使用其中的任何模块来建立自己的动态系统模型并进行动态仿真与分析。Simulink 允许用户建立自己的模块库，将用户定义的一系列的模块或其它库中的相关模块放置在其中，以对自定义模块进行有效的组织与管理。自定义模块库的使用方法与 Simulink 本身提供的模块库使用完全一致。在作进一步的介绍之前，首先说明以下几个常用的术语：

(1) 模块库：具有某种属性的一类模块的集合。

(2) 库模块：模块库中的一个模块。

(3) 引用块：模块库中的一个模块的副本（从模块库中拖动或复制到系统模型中的模块）。

(4) 关联：引用块与对应的模块库中的模块之间的联系，当模块库中的模块发生改变时 Simulink 会自动更新相应的引用块。

注意：每个从模块库中复制或拖动的模块（引用块）都与模块库中的原始模块（库模块）存在着关联。关联的目的有两点：一旦模块库中的模块被修改，其相应的引用块就会按照同样的方式被修改；在没有断开关联之前不能够对引用块进行任何修改。Simulink 可以在模型中每个与模块库之间存在关联的引用块的左下角显示一个箭头。如果想要显示与模块库之间的关联的话，则选择 Format 菜单下的 Library Link Display。

7.4.2　建立与使用模块库

建立 Simulink 模块库的方法如下：

(1) 在 Simulink 中使用 File 菜单下的 New\Library 建立一个新的模块库。

(2) 将用户自定义的模块或是其它模块库中的模块移动到新的模块库中。

(3) 保存新的模块库。

注意：在使用新的模块库之前必须保存新的模块库，然后才能够正确使用其中的模块；一旦模块库被保存之后，其使用方法就与 Simulink 中的模块库的使用方法完全一致。

下面举例说明。首先建立一个新的模块库（其名称为 MyOwnLib），然后将在 7.3 节中封装后的人口动态系统模块移动到新库之中并保存，最后使用此模块库中的模块建立如图 7.28 所示的动态系统。

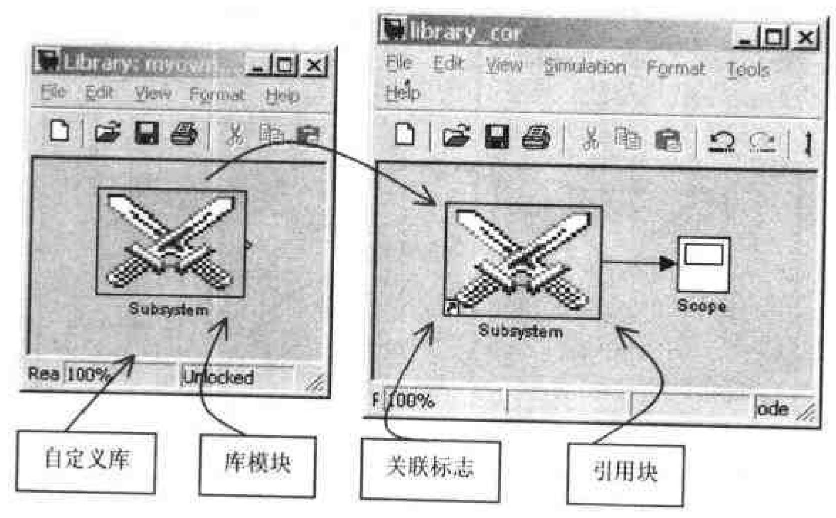


图 7.28　使用自定义模块库建立动态系统

此时双击系统模型中的引用块，则会弹出与 7.3 节中相同的参数设置对话框，只要用户输入正确的参数并设置合适的仿真参数，其仿真结果就会与 7.3 节中的完全一致。

当一个新建立的模块库关闭之后，模块库便被锁住了。用户可以使用 Edit 菜单下的 Look under mask 来查看模块库中选定模块的内部实现。如果需要改变模块库中的内容，则必须使用 Edit 菜单下的 Unlock library 命令（或按下 Unlock 按钮）对模块库进行解锁，或

移动模块库中的模块以弹出如图 7.29 所示的模块库修改确认对话框。

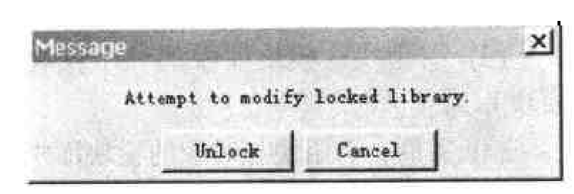


图 7.29 模块库修改确认对话框

如果需要对引用块的内容进行修改，可采用如下两种方法：

(1) 用鼠标右键单击块，选择 Link Options 下的 Go to library block 命令，对与引用块相应的模块库进行解锁，修改模块库中的模块。然后使用 Edit 菜单下的 Update diagram 以更新模型框图，但是这会影响到所有引用块。

(2) 用鼠标右键单击块，选择 Link Options 下的 Disable link 命令（这时关联标志箭头将变为灰色）。然后用鼠标右键单击模块，选择 Look under mask，打开模块以修改模型中模块的内容。这只影响当前的块。

7.4.3 库模块与引用块的关联

Simulink 通过关联来表示模块库中的模块与相应的引用块之间的关系。为了使用户对关联有一个更深入的了解，这里以一个简单的例子来说明关联是如何工作的。

【例 7.4】 关联的使用。其步骤如下：

(1) 建立一个新的系统模型，从自定义的模块库 MyOwnLib 中拖动自定义的系统模块（人口变化动态系统）到此模型框图中。图 7.30 所示为在新建立的系统模型中同时使用两个引用块。

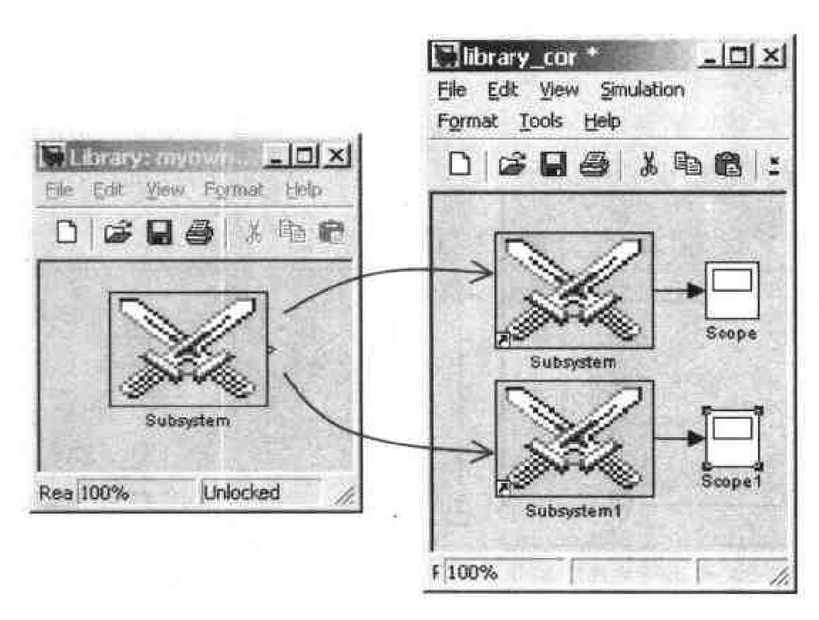


图 7.30 在系统模型框图中使用两个引用块

(2) 断开第二个引用块与库模块的关联：用鼠标右键单击引用块，选择 Link Options 下的 Disable link 命令即可。

(3) 修改模块库中的模块，在子系统中加入注释文本“This is the modified library model”。用鼠标右键单击并选择 Look under mask 命令以修改，如图 7.31 所示。

(4) 使用 Edit 下的 Update diagram 刷新系统模型框图，然后使用 Look under mask 查看引用块的变化，则第一个模块的内容随模块库的改变而改变，而第二个模块并没发生任何

变化，如图 7.32 所示。

(5) 修改禁止关联的第二个引用块，在其中加入注释“Disable Link”，然后保存系统模型并试图恢复关联。用鼠标右键单击第二个引用块，选择 Link options 下的 Restore link 命令，此时 Simulink 将弹出如图 7.33 所示的恢复连接对话框。

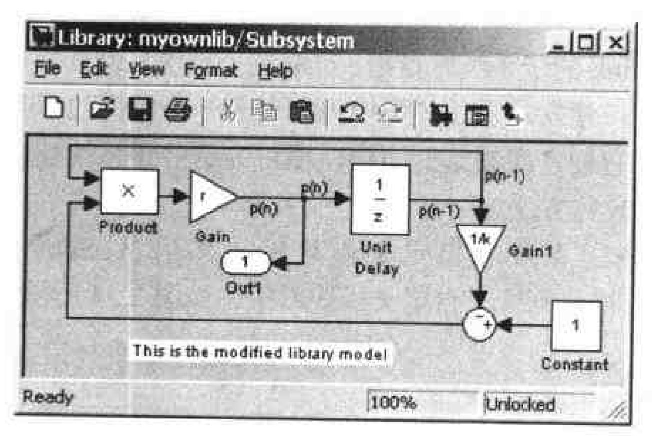


图 7.31　对模块库中的模块进行修改

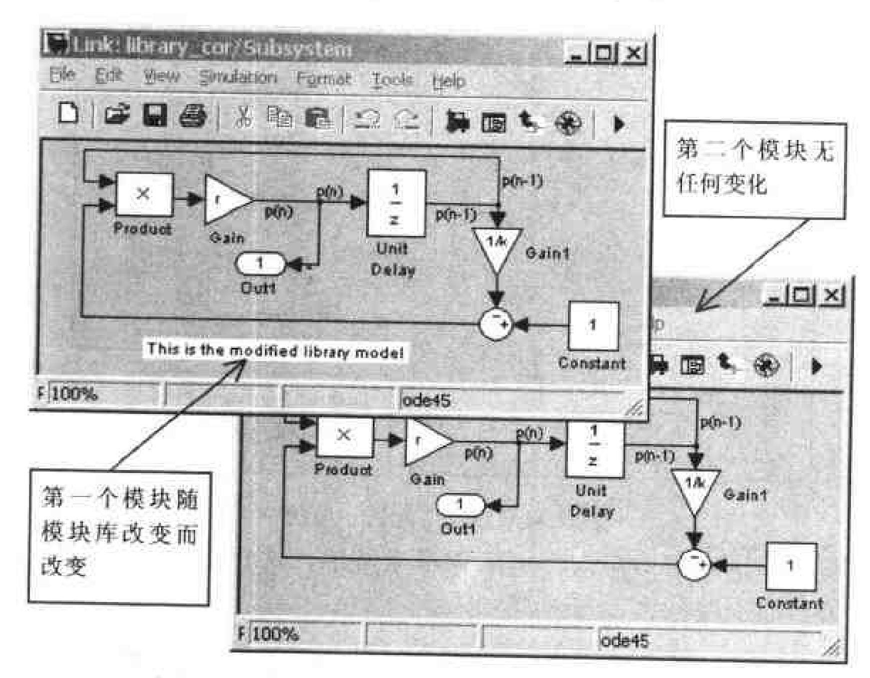


图 7.32　修改模块库中模块对不同引用块的影响

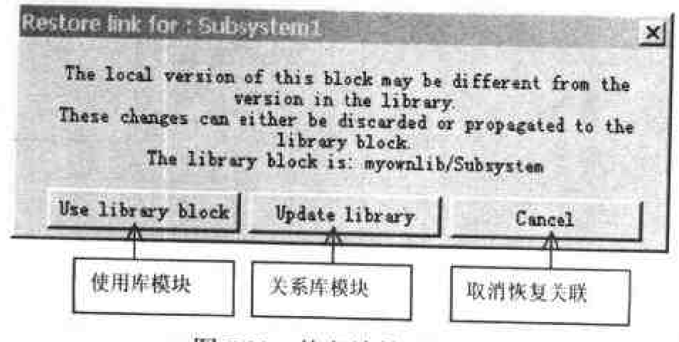


图 7.33　恢复连接对话框

此对话框表示禁止关联的引用块在恢复关联时有可能已经被修改，用户可以选择 Use library block 放弃引用块中的修改使用模块库中的模块，或选择 Update library 更新模块库中的模块（这会影响到其它的使用关联的引用块）。

7.4.4 可配置子系统

7.2 节中介绍 Simulink 的子系统时指出，可配置子系统只能够在用户自定义的模块库使用。在某些情况下，用户建立的系统模型中可能有若干个不同的子系统，它们具有同样的功能。如果用户需要在它们之间频繁的切换，便可为这些子系统建立可配置子系统。建立与使用可配置子系统的具体方法如下：

(1) 建立包含这些子系统的自定义模块库，然后从 Subsystems 模块库中拖动 Configurable Subsystem 模块到这个自定义模块库中，如图 7.34 所示。

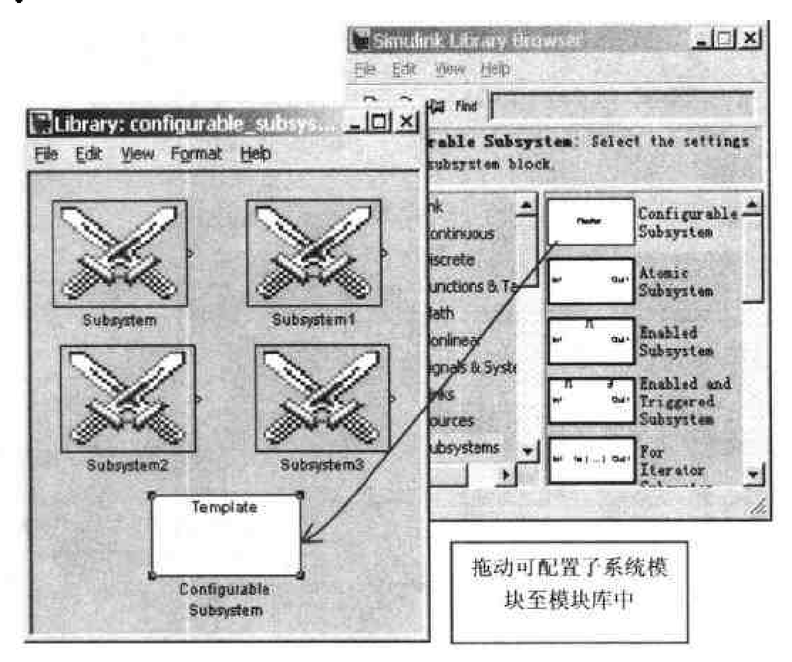


图 7.34 建立可配置子系统

(2) 保存自定义模块库，然后双击可配置子系统块。选择用户需要相互切换的子系统，如图 7.35 所示。

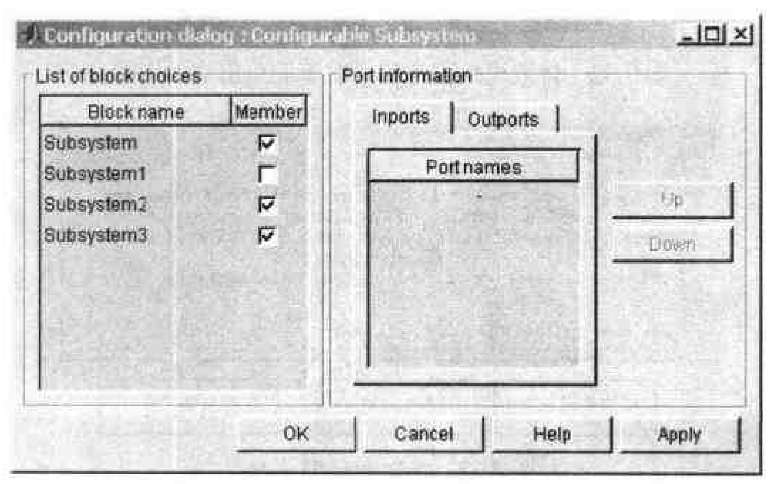


图 7.35 选择需要使用的子系统

(3) 复制模块库中的可配置子系统到相应的系统模型中，如图 7.36 所示。

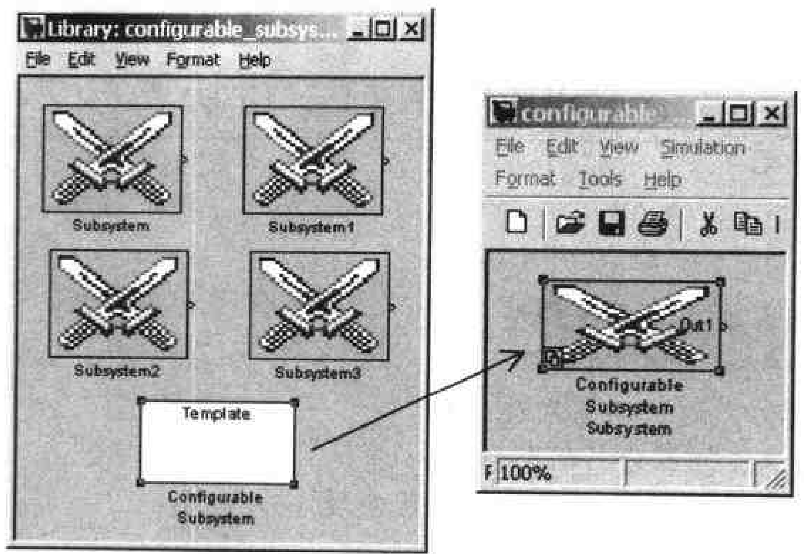
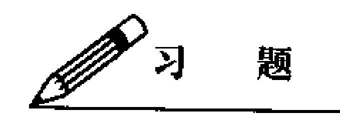


图 7.36　在系统模型中使用可配置子系统

(4) 使用 Edit 下的 Block Choice 项完成模块的配置。

此外，模块库中的模块可能具有不同的输入和输出端口，但它们可以在可配置子系统中被合并。限于篇幅，这里不再赘述。总的来说，Simulink 的模块库技术为用户提供了足够的功能来很好地组织管理自定义的模块。对于不同领域的用户可以建立自己独有的模块库，从而使得系统的设计与开发具有可继承性，便于系统的维护与修改。



习　题

1. 由单位正弦信号产生单位方波信号。要求：使用使能子系统，且单位正弦信号作为使能子系统的控制信号。

提示：建立一使能子系统，当正弦信号为正时系统输出为 1，当正弦信号为负时系统输出为-1。

2. 封装第 5 章中所建立的蹦极系统。要求：封装后的蹦极子系统只有一个输出端口，封装后子系统的参数设置包括蹦极者的体重、弹性绳索的弹性常数。通过仿真分析蹦极系统在下述情况下是否安全，并绘制相应的响应曲线：

(1) 蹦极者体重 80 kg，弹力绳索的弹性常数为 30。

(2) 蹦极者体重 70 kg，弹力绳索的弹性常数为 20。

第8章 Simulink 命令行仿真技术

内容概要

- 使用命令行方式建立系统模型
- Simulink 与 MATLAB 的接口
- 使用命令行方式进行动态系统仿真
- 使用 MATLAB 脚本分析动态系统
- 确定系统状态、求取平衡点与线性化处理
- MATLAB 回调函数

在前面各个章节的介绍之中，动态系统模型的建立、系统的仿真与分析等，均是使用 Simulink 的图形建模方式实现的。Simulink 的图形建模方式给用户提供了强大的功能与友好的使用界面，使用 Simulink 的图形建模方式就可以完成绝大多数的动态系统仿真分析。这对于一般的用户而言已经足够了，但是对于一些高级的系统设计分析者而言还是不够的。这是因为，虽然 Simulink 的图形建模方式非常友好，使用方便而且功能也非常强大，但是 Simulink 的图形建模方式在有的时候限制了用户对系统模型更深程度的操作以及对系统仿真做更多的控制与修改。例如，在系统仿真过程中需要对系统中模块的参数进行修改以满足特定的要求，或需要分析系统在不同输入信号下的响应等等。

本章将介绍 Simulink 的命令行仿真技术。首先给出命令行仿真的基本概念，命令行仿真指的是在动态系统设计、模型建立、仿真与分析之中，使用 MATLAB 命令行的方式对系统的仿真分析做更多的操作与控制，而非仅仅使用 Simulink 的图形建模方式进行控制。与图形建模方式操作控制相比，命令行方式仿真具有如下优点：

(1) 自动地重复运行仿真。

(2) 在仿真过程中动态调整参数。

(3) 分析不同输入信号下的系统响应。

(4) 进行快速仿真。

其实在第 4 章介绍的 Simulink 与 MATLAB 的接口设计中，已经使用了命令行仿真技术。例如，从 MATLAB 的工作空间向 Simulink 模型传递模块参数、输入信号以及将 Simulink 的仿真结果输出到 MATLAB 的工作空间之中等均可视为命令行仿真技术的使用。本章将对 Simulink 的命令行仿真技术进行更为详细的介绍。

8.1　使用命令行方式建立系统模型

除了使用 Simulink 的图形建模方式建立动态系统模型之外，用户也可以使用命令行方式进行系统建模，然后再进行动态系统的仿真与分析。在进一步介绍使用命令行进行动态系统的仿真技术之前，首先简单介绍一下使用命令行的方式建立系统模型的相关知识。Simulink 中建立系统模型的命令如表 8.1 所示。

表 8.1　系统模型建立命令

命　令	功　能
new_system	建立一个新的 Simulink 系统模型
open_system	打开一个已存在的 Simulink 系统模型
close_system, bdclose	关闭一个 Simulink 系统模型
save_system	保存一个 Simulink 系统模型
find_system	查找 Simulink 系统模型、模块、连线及注释
add_block	在系统模型中加入指定模块
delete_block	从系统模型中删除指定模块
replace_block	替代系统模型中的指定模块
add_line	在系统模型中加入指定连线
delete_line	从系统模型中删除指定连线
get_param	获取系统模型中的参数
set_param	设置系统模型中的参数
gcb	获得当前模块的路径名
gcs	获得当前系统模型的路径名
gcbh	获得当前模块的操作句柄
bdroot	获得最上层系统模型的名称
simulink	打开 Simulink 的模块库浏览器

使用上面的命令便可以生成和编辑动态系统的 Simulink 模型，由于使用命令行方式建立的 Simulink 系统模型与使用图形建模方式建立的系统模型没有什么大的分别，因此这里仅简单介绍各个命令的使用，而不再给出使用这些命令所建立的系统模型框图。

1. new_system

1) 使用语法

```
new_system('sys')
```

2) 功能描述

使用给定的名称建立一个新的 Simulink 系统模型。如果'sys'为一个路径，则新建立的系统为在此路径中指定的系统模型下的一个子系统。注意，new_system 命令并不打开系统模型窗口。

3) 举例

```
new_system('mysys')          % 建立名为'mysys'的系统模型
new_system('vdp/mysys')      % 建立系统模型 vdp 下的子系统'mysys'
```

2. open_system

1) 使用语法

```
open_system('sys')
open_system('blk')
open_system('blk', 'force')
```

2) 功能描述

打开一个已存在的 Simulink 系统模型。

open_system('sys')：打开名为'sys'的系统模型窗口或子系统模型窗口。注意，这里'sys'使用的是 MATLAB 中标准路径名（绝对路径名或相对于已经打开的系统模型的相对路径名）。

open_system('blk')：打开与指定模块'blk'相关的对话框。

open_system('blk', 'force')：打开封装后的子系统，这里'blk'为封装子系统模块的路径名。这个命令与图形建模方式中的 Look under mask 菜单功能一致。

3) 举例

```
open_system('controller')          % 打开名为 controller 的系统模型
open_system('controller/Gain')     % 打开 controller 模型下的增益模块 Gain 的对话框
```

3. save_system

1) 使用语法

```
save_system
save_system('sys')
save_system('sys', 'newname')
```

2) 功能描述

保存一个 Simulink 系统模型。

save_system：使用当前名称保存当前顶层的系统模型。

save_system('sys')：保存已经打开的系统模型，与 save_system 功能类似。

save_system('sys', 'newname')：使用新的名称 newname 保存当前已经打开的系统模型。

3) 举例

```
save_system                      % 保存当前的系统模型
save_system('vdp')               % 保存系统模型 vdp
save_system('vdp', 'myvdp')      % 保存系统模型 vdp，模型文件名为 myvdp
```

4. close_system, bdclose

1) 使用语法

```
close_system
close_system('sys')
close_system('sys', saveflag)
close_system('sys', 'newname')
close_system('blk')
bdclose; bdclose('sys'); bdclose('all')
```

2) 功能描述

关闭一个 Simulink 系统模型。

close_system：关闭当前系统或子系统模型窗口。如果顶层系统模型被改变，系统会提示是否保存系统模型。

close_system('sys')：关闭指定的系统或子系统模型窗口。

close_system('sys', saveflag)：关闭指定的顶层系统模型窗口并且从内存中清除。saveflag 为 0 表示不保存系统模型，为 1 表示使用当前名称保存系统模型。

close_system('sys', 'newname')：将指定的系统模型保存至新的模型文件中并关闭系统。

close_system('blk')：关闭与模块'blk'相关联的对话框。

bdclose、bdclose('sys')、bdclose('all')：无条件地关闭所有系统模型。

3) 举例

```
close_system                    % 关闭当前系统
close_system('engine', 1)       % 保存当前系统模型 engine（使用当前系统名称），然后再关闭系统
close_system('engine/Combustion/Unit Delay')
% 关闭系统模型 engine 下的 Combustion 子系统中 Unit Delay 模块的对话框
```

5. find_system

1) 使用语法

```
find_system(sys, 'c1', cv1, 'c2', cv2,...'p1', v1, 'p2', v2,...)
```

2) 功能描述

查找由 sys 指定的系统模型、模块、连线及注释等等，并返回相应的路径名与操作句柄。由于使用此命令涉及较多的参数设置，因此这里不再赘述，用户可以查看 Simulink 的联机帮助系统中 Simulink 目录下的 Using Simulink\Model Construction Commands\Introduction 中的 find_system 命令的帮助即可。

6. add_block

1) 使用语法

```
add_block('src', 'dest')
add_block('src', 'dest', 'parameter1', value1, ...)
```

2) 功能描述

在系统模型中加入指定模块。

add_block('src', 'dest')：拷贝模块'src'为'dest'（使用路径名表示），从而可以从 Simulink 的模块库中复制模块至指定系统模型中，且模块'dest'参数与'src'完全一致。

add_block('src', 'dest_obj', 'parameter1', value1, ...)：功能与上述命令类似，但是需要设置给定模块的参数'parameter1'，value1 为参数值。

3) 举例

```
add_block('simulink3/Sinks/Scope', 'engine/timing/Scope1')
  % 从 Simulink 的模块库 Sinks 中复制 Scope 模块至系统模型 engine 中子系统 timing 中，其名称
  % 为 Scope1
```

7. delete_block

1) 使用语法

```
delete_block('blk')
```

2) 功能描述

从系统模型中删除指定模块。

delete_block('blk')：从系统模型中删除指定的系统模块'blk'。

3) 举例

```
delete_block('vdp/Out1')          %从 vdp 模型中删除模块 Out1
```

8. replace_block

1) 使用语法

```
replace_block('sys', 'blk1', 'blk2', 'noprompt')
replace_block('sys', 'Parameter', 'value', 'blk', ...)
```

2) 功能描述

替代系统模型中的指定模块。

replace_block('sys', 'blk1', 'blk2')：在系统模型'sys'使用模块'blk2'取代所有的模块'blk1'。如果'blk2'为 Simulink 的内置模块，则只需要给出模块的名称即可，如果为其它的模块，必须给出所有的参数。如果省略'noprompt'，Simulink 会显示取代模块对话框。

replace_block('sys', 'Parameter', 'value', ..., 'blk')：取代模型'sys'中具有特定取值的所有模块'blk'。'Parameter'为模块参数，'value'为模块参数取值。

3) 举例

```
replace_block('vdp','Gain','Integrator','noprompt')
    % 使用积分模块 Integrator 取代系统模型 vdp 中所有的增益模块 Gain，并且不显示取代对话框
```

9. add_line、delete_line

1) 使用语法

```
h = add_line('sys','oport','iport')
h = add_line('sys','oport','iport', 'autorouting','on')
delete_line('sys', 'oport', 'iport')
```

2) 功能描述

在系统模型中加入或删除指定连线。

add_line('sys', 'oport', 'iport')：在系统模型'sys'中给定模块的输出端口与指定模块的输入端口之间加入直线。'oport'与'iport'分别为输出端口与输入端口（包括模块的名称、模块端口编号）。

add_line('sys','oport','iport', 'autorouting','on')：与 add_line('sys','oport','iport')命令类似，只是加入的连线方式可以由'autorouting'的状态控制：'on'表示连线环绕模块，而'off'表示连线为直线（缺省状态）。

delete_line('sys', 'oport', 'iport')：删除由给定模块的输出端口'oport'至指定模块的输入端口'iport'之间的连线。

3) 举例

```
add_line('mymodel','Sine Wave/1','Mux/1')
```

```
% 在系统模型'mymodel'中加入由正弦模块Sine Wave的输出至信号组合模块Mux第一个输入间
% 的连线
delete_line('mymodel', 'Sine Wave/1','Mux/1')
% 删除系统模型'mymodel'中由正弦模块Sine Wave的输出至信号组合模块Mux第一个输入间的
% 连线
```

10. set_param 和 get_param

1) 使用语法

```
set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)
get_param('obj', 'parameter')
get_param( { objects }, 'parameter')
get_param(handle, 'parameter')
get_param(0, 'parameter')
get_param('obj', 'ObjectParameters')
get_param('obj', 'DialogParameters')
```

2) 功能描述

设置与获得系统模型以及模块参数。

set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)：其中'obj'表示系统模型或其中的系统模块的路径，或者取值为 0；给指定的参数设置合适的值，取值为 0 表示给指定的参数设置为缺省值。在仿真过程中，使用此命令可以在 MATLAB 的工作空间中改变这些参数的取值，从而可以更新系统在不同的参数下运行仿真。

get_param('obj', 'parameter')：返回指定参数取值。其中'obj'为系统模型或系统模型中的系统模块。

get_param({ objects }, 'parameter')：返回多个模块指定参数的取值，其中{ objects }表示模块的细胞矩阵（Cell）。

get_param(handle, 'parameter')：返回句柄值为 handle 的对象的指定参数的取值。

get_param(0, 'parameter')：返回 Simulink 当前的仿真参数或默认模型、模块的指定参数的取值。

get_param('obj', 'ObjectParameters')：返回描述某一对象参数取值的结构体。其中返回到结构体中具有相应的参数名称的每一个参数域分别包括参数名称（如 Gain）、数据类型，如 string 以及参数属性如 read-only 等。

get_param('obj', 'DialogParameters')：返回指定模块对话框中所包含的参数名称的细胞矩阵。

由于参数选项很多，这里不再赘述,读者可以参考 Simulink 帮助中的 Model and BlockParameters 中的详细介绍。

3) 举例

```
set_param('vdp', 'Solver', 'ode15s', 'StopTime', '3000')  % 设置系统模型'vdp'的求解器为'ode15s'，仿真结
                                                          % 束时间为 3000 s
set_param('vdp/Mu', 'Gain', '1000')  % 设置系统模型 vdp 中模块 Mu 中的增益模块 Gain 的取值为 1000
set_param('vdp/Fcn', 'Position', [50 100 110 120])       % 设置系统模型 vdp 中的 Fcn 模块的位置为
```

```
                                        % [50 100 110 120]
set_param('mymodel/Zero-Pole','Zeros','[2 4]','Poles','[1 2 3]')
  % 设置系统模型 mymodel 中零极点模块 Zero-Pole 的零点 Zeros 取值为[2 4]，而极点 Poles 取值为
  % [1 2 3]
get_param('Mysys/Subsys/Inertia','Gain')
  % 获得系统模型 Mysys 中子系统 Subsys 中的 Inertia 模块的增益值。其结果如下（假设此系统模型
  % 以存在）
    ans =
              1.0000e-006
    p = get_param('untitled/Sine Wave', 'DialogParameters')
    % 获得系统模型 untitled 下正弦信号模块 Sine Wave 的参数设置对话框中所包含的参数名称，
    % 结果如下
    p =
        'Amplitude'
        'Frequency'
    'Phase'
        'SampleTime'
```

11. gcb、gcs 以及 gcbh

1) 使用语法

```
gcb
gcb('sys')
gcs
gcbh
bdroot
bdroot('obj')
```

2) 功能描述

gcb：返回当前系统模型中当前模块的路径名。

gcb('sys')：返回指定系统模型中当前模块的路径名。

gcs：获得当前系统模型的路径名。

gcbh：返回当前系统模型中当前模块的操作句柄。

bdroot：返回顶层系统模型的名称。

bdroot('obj')：返回包含指定对象名称的顶层系统的名称，其中'obj'为系统或模块的路径名。

注意：所谓的当前系统模型指的是满足如下条件的系统模型：

(1) 在模型编辑过程中最近点击的系统或子系统。

(2) 在含有 S-函数模块的系统模型仿真过程中，包含任何正在求值的含有 S-函数的模块的系统或子系统。

(3) 在回调过程中，包含任何正在执行的具有回调函数模块的系统或子系统。

(4) 在封装子系统模块的参数初始化中，包含任何正在进行封装参数初始化模块的系统或子系统。

(5) 当前系统模块的概念与当前系统模型相类似。

12. simulink

1) 使用语法

simulink

2) 功能描述

打开 Simulink 的模块库浏览器。

使用命令行方式建立的动态系统模型与使用 Simulink 图形建模方式所建立的系统模型完全一致，然而，仅仅使用命令行方式建立系统模型是一件麻烦且低效的方法。其实在实际的应用中，很少单独使用 Simulink 的命令行方式，它总是与 Simulink 的图形建模方式操作相结合，从而给高级用户提供了对建立动态系统模型进行系统仿真与分析更为灵活的控制。本章以下各节内容便逐一介绍如何使用命令行对动态系统仿真进行更多的控制。

8.2　回顾与复习：Simulink 与 MATLAB 的接口

在第 4 章中介绍 Simulink 与 MATLAB 的接口技术时，已经使用了 Simulink 的命令行方式。当然，在动态系统模型建立的过程中，并没有使用命令行方式，而只是在使用 Simulink 对动态系统仿真分析的过程中应用了命令行方式。在进一步介绍使用命令行方式对动态系统仿真分析进行更多的控制之前，首先以 Simulink 与 MATLAB 之间最简单的接口设计来介绍一下命令行仿真的基础知识。

先建立一个非常简单的动态系统，其功能如下：

(1) 系统的输入为一单位幅值、单位频率的正弦信号。

(2) 系统的输出信号为输入信号的积分。

其要求是：

(1) 系统的输入信号由 MATLAB 工作空间中的变量提供，时间范围为 0 至 10 s。

(2) 使用 MATLAB 绘制原始输入信号与系统运算结果的曲线。

从动态系统的功能与其要求可知，建立此动态系统的模型需要以下一些模块：

(1) Sources 模块库中的 In1 模块：表示系统的输入。

(2) Sinks 模块库中的 Out1 模块：表示系统的输出。

(3) Continuous 模块库中的 Integrator 模块：积分运算器。

其中 In1（Inport）模块与 Out1（Outport）模块都是虚模块，它们仅仅表示将信号传入或传出子系统。当它们在最顶层的系统模型中使用时，可以通过它们从 MATLAB 工作空间中输入信号并将计算结果输出到 MATLAB 工作空间中。图 8.1 所示为此系统的模型框图。

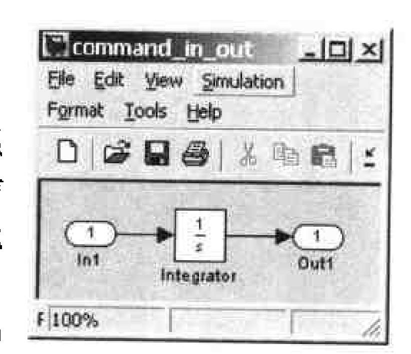


图 8.1　简单动态系统模型框图

为了从 MATLAB 工作空间中输入信号，选择 Simulation Parameters 仿真参数设置对话框中的 Workspace I/O 选项卡，设置外部输入，并给出相应的变量名。图 8.2 所示为 Workspace I/O 选项卡的设置。

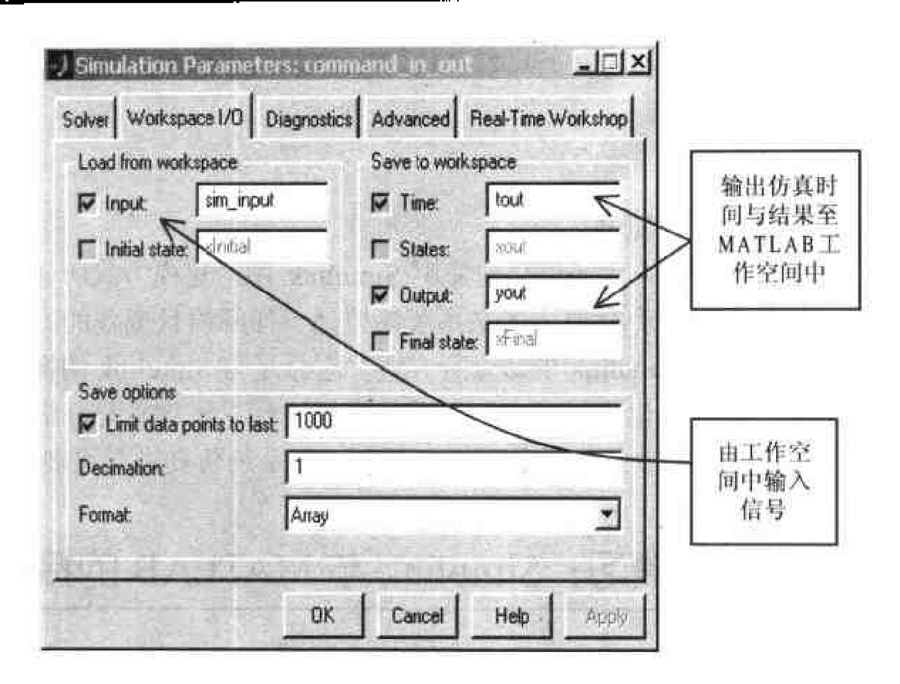


图 8.2　Workspace I/O 设置

然后在 MATLAB 工作空间中定义输入变量 sim_input 如下：

```
>> t=0:0.1:10;                 % 表示输入信号的时间范围
>> u=sin(t);                   % 产生输入正弦信号
>> sim_input=[t',u'];          % 传递至 Simulink 系统模型的变量
```

接下来，采用默认的系统仿真参数并运行系统仿真。最后使用 MATLAB 命令绘制出原始输入信号与系统运算结果，如下所示：

```
>> plot(t,u,tout,yout,'--')    %绘制系统输入信号与仿真结果，如图 8.3 所示
```

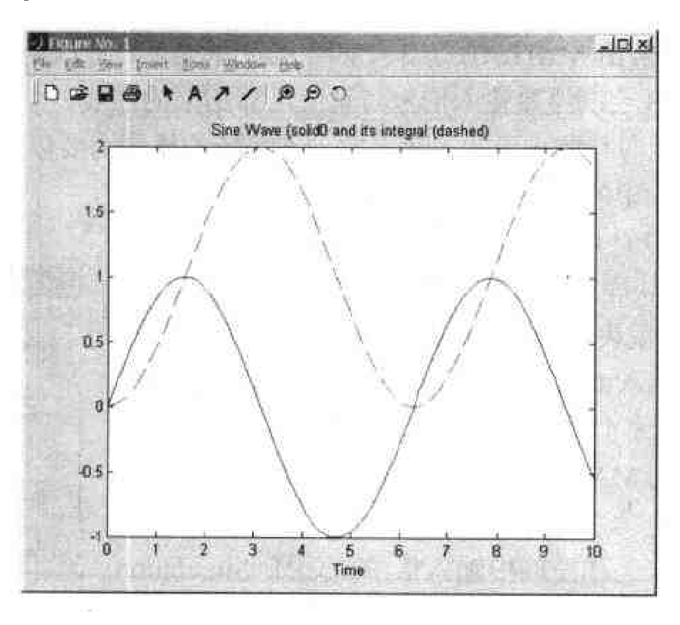


图 8.3　系统原始输入与运算结果曲线

在对此系统进行仿真时，仍然采用在第 5 章中介绍的 Simulink 的图形建模方式操作实现，至于使用命令行的方式对动态系统进行仿真的方法将在 8.3 节中进行介绍。本节只是非

常简单地介绍了使用命令行与 Simulink 图形建模方式操作共同进行动态系统仿真最为基础的知识。

8.3 使用命令行方式进行动态系统仿真

用户可能会有这样的疑问：既然采用 Simulink 的图形建模方式已经能够进行动态系统仿真了，为何还需要使用命令行方式对动态系统进行仿真呢？这是因为，使用命令行方式，用户可以编写并运行系统仿真的脚本文件来完成动态系统的仿真，在脚本文件中重复地对同一系统在不同的仿真参数或不同的系统模块参数下进行仿真，而无需一次又一次启动 Simulink 图形窗口中的 Start Simulation 进行仿真；如果需要分析某一参数对系统仿真结果的影响，用户可以很容易地通过 for 循环自动修改任意指定的参数即可。这样可以非常容易地分析不同参数对系统性能的影响，并且也可以从整体上加快系统仿真的速度。使用命令行方式进行动态系统的仿真给用户提供了更强的控制能力，尤其适合高级用户。下面对进行动态系统仿真的命令逐一介绍。

8.3.1 使用 sim 命令进行动态系统仿真

1. 使用语法

```
[t,x,y] = sim(model, timespan, options, ut)
[t,x,y1, y2, ..., yn] = sim(model, timespan, options, ut);
```

以上是完整的语法格式，实际使用中可以省略其中的某些参数设置而采用默认参数进行仿真。

2. 功能描述

对指定的系统模型按照给定的仿真参数与模型参数进行系统仿真。仿真所使用的参数包括所有使用仿真参数对话框的设置、MATLAB 工作空间的输入输出选项卡中的设置以及采用命令行方式设置的仿真参数与系统模块参数。

除参数'model'外，其它的仿真参数设置均可以取值为空矩阵，此时 sim 命令对没有设置的仿真参数使用默认的参数值进行仿真。默认的参数值由系统模型框图所决定。用户可以使用 sim 命令的 options 参数对可选参数进行设置，这样设置的仿真参数将覆盖模型默认的参数。

如果用户对连续系统进行仿真，必须设置合适的仿真求解器，因为默认的仿真求解器为变步长离散求解器（Variable Step Discrete Solver）。可以使用 simset 命令进行设置（后面将简单介绍 simset 命令的使用）。

3. 参数说明

(1) model：需要进行仿真的系统模型框图名称。

(2) timespan：系统仿真时间范围（起始时间至终止时间），可以为如下的形式：

① tFinal：设置仿真终止时间。仿真起始时间默认为 0。

② [tStart tFinal]：设置仿真起始时间（tStart）与终止时间（tFinal）。

③ [tStart OutputTimes tFinal]：设置仿真起始时间（tStart）与终止时间（tFinal），并

且设置仿真返回的时间向量[tStart OutputTimes tFinal]，其中 tStart、OutputTimes、tFinal 必须按照升序排列。

(3) options：由 simset 命令所设置的除仿真时间外的仿真参数（为一结构体变量）。

(4) ut：表示系统模型顶层的外部可选输入。ut 可以是 MATLAB 函数。可以使用多个外部输入 ut1、ut2…。其格式必须符合输入信号的要求（可参考第 4 章 Simulink 与 MATLAB 的接口设计中的相关内容——由 MATLAB 工作空间传递信号至系统模型）。

(5) t：返回系统仿真时间向量。

(6) x：返回系统仿真状态变量矩阵。首先是连续状态，然后是离散状态。

(7) y：返回系统仿真的输出矩阵。按照顶层输出 Outport 模块的顺序输出，如果输出信号为向量输出，则输出信号具有与此向量相同的维数。

(8) y1, ... ,yn：返回多个系统仿真的输出。

8.3.2 举例之一：简单仿真

在 8.2 节中以动态系统 command_in_out 为例，对 Simulink 与 MATLAB 接口设计作了简单的回顾与复习。Simulink 系统模型的输入信号为 MATLAB 工作空间中的变量，同时仿真结果也被输出到 MATLAB 工作空间中。但在进行系统仿真时，并没有使用命令行方式。实际上用户也可以使用命令行方式对此系统模型进行仿真，在仿真之前，首先使用仿真参数设置对话框设置参数。然后在 MATLAB 命令窗口中键入如下命令进行系统模型的仿真，并绘制此系统的输入信号与仿真结果：

```
>> t=0:0.1:10;
>> u=sin(t);
>> sim_input=[t',u'];
>> [tout, x, yout]=sim('command_in_out');
% 使用命令行 sim 进行系统仿真，仿真参数采用与 8.2 节中相同的仿真参数
>> plot(t,u,tout,yout,'--')
```

在对动态系统 command_in_out 进行仿真时，必须先打开此系统模型并且设置合适的仿真参数。sim 命令的参数为模型的名称（由不带扩展名.mdl 的系统模型文件名构成）。为突出 sim 命令的作用，这里仍使用 Simulink 的仿真参数设置对话框对仿真参数设置，至于使用 simset 命令设置仿真参数将在后面进行介绍。使用 sim 命令进行系统仿真前后 MATLAB 工作空间中的变量列表如图 8.4 所示。

图 8.4 中左侧为使用 sim 命令进行仿真之前的 MATLAB 工作空间变量列表（其中 sim_input 为系统模型的输入信号、t 为输入信号的时间范围、u 为输入信号的取值）；右侧为使用 sim 命令进行仿真之后的工作空间变量列表（其中 tout 为输出时间变量、x 为输出系统状态变量、yout 为系统运算结果；如果在系统模型的顶层没有 Outport 模块，则输出 yout 为空）。从图中可以明显看出，动态系统的仿真结果（包括输出时间向量、系统状态与运算结果）被输出到了 MATLAB 工作空间中。此外，由 plot 所绘制的图形也与 8.2 节中的绘制结果完全一致，这里不再给出。

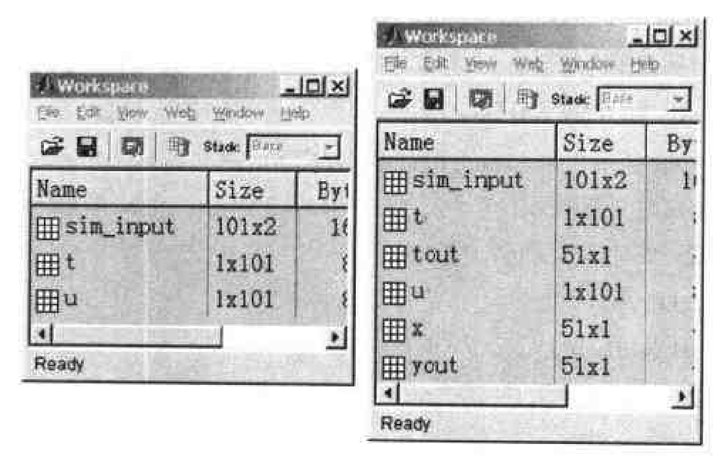


图 8.4　系统仿真前后 MATLAB 工作空间变量列表

8.3.3　举例之二：仿真时间设置

在前面已经对 sim 命令中的仿真时间参数 timespan 设置做了介绍。timespan 具有三种使用形式，根据不同动态系统仿真的不同要求，用户可以选择使用如下所示的不同形式进行系统仿真：

```
>>[t,x,y]=sim(model,tFinal)
>>[t,x,y]=sim(model,[tStart tFinal])
>>[t,x,y]=sim(model,[tStart outputTimes tFinal])
```

注意，其中系统仿真结束时间 tFinal 应该大于系统仿真开始时间 tStart。此外，在默认情况下，系统仿真的输出结果（输出时间、状态及系统运算结果）受到 Simulink 求解器仿真步长的控制。由于仿真时刻受到求解器仿真步长的控制，因而系统仿真输出结果也受到求解器步长的控制。如果需要在指定的时刻输出系统仿真结果，则需要使用仿真时间设置的第三种方式，其中[tStart outputTimes tFinal]表示输出时间向量，此向量为一递增行向量。在 sim 命令中使用 timespan 设置系统仿真时间范围，会覆盖 Simulink 原来的仿真时间设置，但是并不会影响到系统模型。

仍以前面的动态系统 command_in_out 为例说明使用命令行方式时如何对系统仿真时间进行设置。首先打开系统模型 command_in_out，然后仍然使用 Simulink 仿真参数设置对话框对系统的仿真参数进行设置，其设置与 8.2 节中所使用的参数完全一致。为了使用户对使用命令行方式设置系统仿真时间范围有一个直观的了解，这里使用四组不同的仿真时间对此系统进行仿真，并绘制系统输入信号与系统运算结果关系图以作比较。在 MATLAB 命令窗口中运行如下的命令：

```
>> t=0:0.1:10;
>> u=sin(t);
>> sim_input=[t',u'];
>> [tout1, x1, yout1]=sim('command_in_out', 5);
% 系统仿真时间范围为 0 至 5 s，输出时间向量 tout1 由 Simulink 的求解器步长变化决定
>> [tout2, x2, yout2]=sim('command_in_out', [1 8]);
% 系统仿真时间范围为 1 至 8 s，输出时间向量 tout1 同样由 Simulink 的求解器步长变化决定
```

```
>> [tout3, x3, yout3]=sim('command_in_out', 1:8);
% 系统仿真时间范围为 1 至 8 s，并且每隔 1 s 输出一次，即输出时间变量为
% [1  2  3  4  5  6  7  8]
>> [tout4, x4, yout4]=sim('command_in_out', 1:0.2:8);
    % 系统仿真时间范围为 1 至 8 s，并且每隔 0.2 s 输出一次，即输出时间变量为
    % [1  1.2  %  1.4 … 7.6  7.8  8]
>> subplot(2,2,1)
>> plot(t,u,tout1,yout1,'.');
>> subplot(2,2,2)
>> plot(t,u,tout2,yout2,'.');
>> subplot(2,2,3)
>> plot(t,u,tout3,yout3,'.');
>> subplot(2,2,4)
>> plot(t,u,tout4,yout4,'.');
% 在同一幅图中绘制系统 command_in_out 的输入信号以及在不同仿真时间范围上的系统运算结
% 果，其结果如图 8.5 所示
```

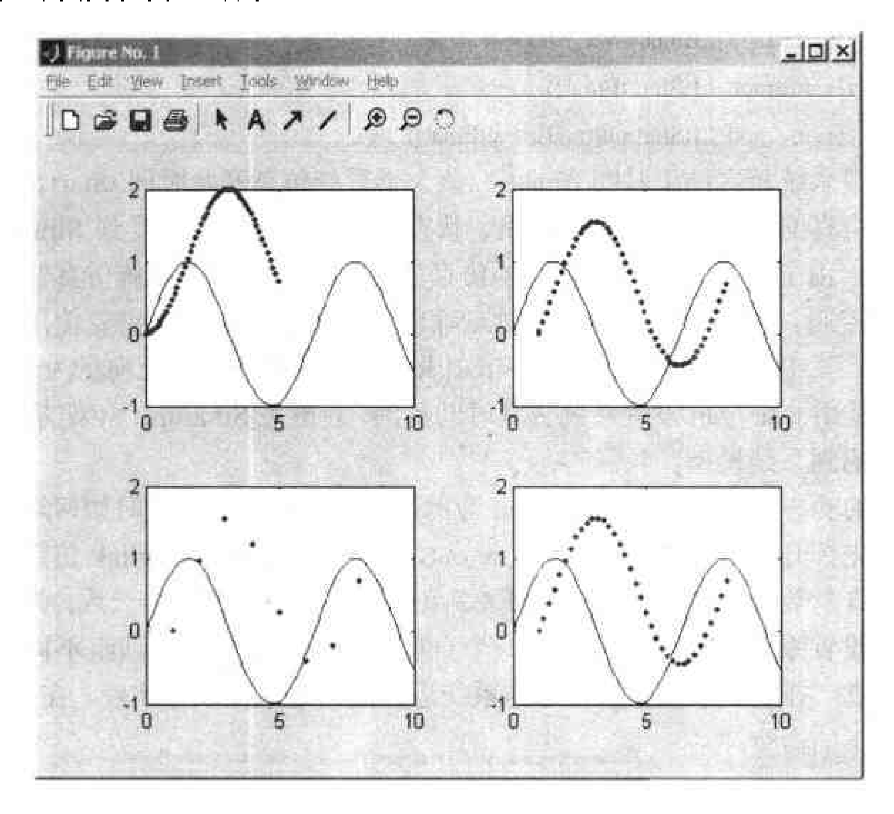


图 8.5 在不同仿真时间设置下的系统仿真结果比较

说明：图中实线所示为系统的输入信号，圆点所代表的数据为系统的输出信号，使用圆点进行绘制图形可以很直观地观察系统仿真所返回的时间向量。其中，左上图所示系统仿真时间范围为 0 至 5 s，系统仿真所返回的时间向量的大小（即系统仿真输出结果的点数）由 Simulink 的求解器步长控制；右上图所示系统仿真时间范围为 0 至 8 s（可以从图中明显看出），系统仿真所返回的时间向量大小同样受到 Simulink 求解器步长的控制；左下图所示

系统仿真时间范围仍为 0 至 8 s，但是此时系统仿真返回到的时间向量为[1 2 3 4 5 6 7 8]，故系统的输出结果仅有 8 个数据点；右下图所示系统仿真时间范围仍为 0 至 8 s，但是此时系统仿真返回的时间向量为[1 1.2 1.4 ... 7.6 7.8 8]，故系统的输出结果具有 36 个数据点。从 MATLAB 的工作空间的变量列表中可以查看不同仿真时间设置下，系统仿真所返回的时间向量输出，如图 8.6 所示。

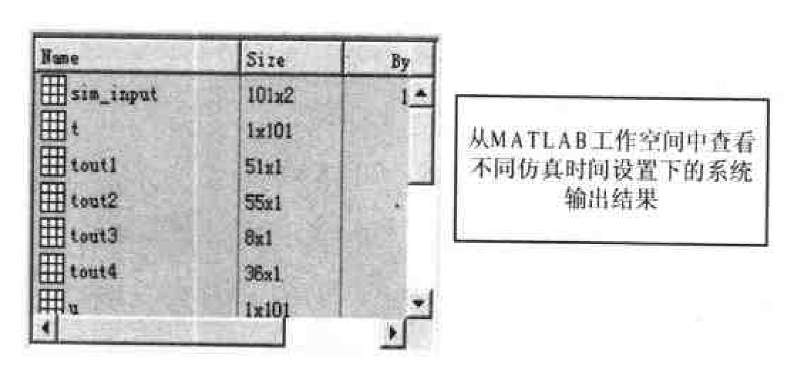


图 8.6　系统仿真返回的时间向量输出

由此可见，使用 sim 可以对动态系统的仿真时间进行自由的设置，以对动态系统的仿真与分析做更多的控制。

8.3.4　举例之三：外部输入变量设置

前面对动态系统 command_in_out 进行仿真时，通过设置 Simulink 仿真参数设置对话框中 Workspace I/O 中的外部变量输入，以使系统在仿真过程中从 MATLAB 的工作空间中获取输入信号 sim_input。除了使用这种方法从 MATLAB 工作空间中获得系统输入信号之外，用户还可以通过使用 sim 命令中的 ut 参数来设置系统的外部输入信号。下面介绍如何使用 ut 参数设置外部输入信号。

1. ut 参数的生成

用户可以使用命令 [t,x,y]=sim(model, timespan, options, ut) 对动态系统进行仿真并且从 MATLAB 工作空间中输入变量。其中 ut 为一个具有两列的矩阵，第一列表示外部输入信号的时刻，第二列表示与给定时刻相应的信号取值。使用矩阵 ut 能够为系统模型最顶层的 Inport 模块提供外部输入，并将自动覆盖 Simulink 仿真参数设置对话框中 Workspace I/O 中的设置。此外，当输入信号中存在着陡沿边缘时，必须在同一时刻处定义不同的信号取值。例如，对于图 8.7 所示的一个类似于方波的信号。

产生此输入信号的 MATLAB 命令为

```
>>ut=[0 1;10 1;10 -1;20 -1;20 1;30 1;30 -1;40 -1;40 1;50 1]
```

2. 应用举例

仍以动态系统 command_in_out 为例说明如何使用 sim 命令的参数 ut 从 MATLAB 工作空间中获得输入信号。在 MATLAB 命令窗口中键入如下的命令：

```
>> t=0:0.1:10;
>> u=sin(t);
>> sim_input=[t',u'];
>> [tout1, x1, yout1]=sim('command_in_out', 10);
```

```
% 使用 Simulink 仿真参数对话框中的 Workspace I/O 从 MATLAB 工作空间中获得输入信号
>>u=cos(t);
>>ut=[t',u'];              % 改变系统输入信号
>>[tout2,x2,yout2]=sim('command_in_out',10,[],ut);
% 使用 sim 命令的 ut 参数获得系统输入信号，ut 的使用会覆盖由 Workspace I/O 的系统输入设置.
% 这一点可以在下面的系统仿真结果图形中反映出来
>>subplot(1,2,1); plot(tout1,yout1);
>>subplot(1,2,2);plot(tout2,yout2);
%绘制系统在不同输入信号下的响应曲线，如图 8.8 所示
```

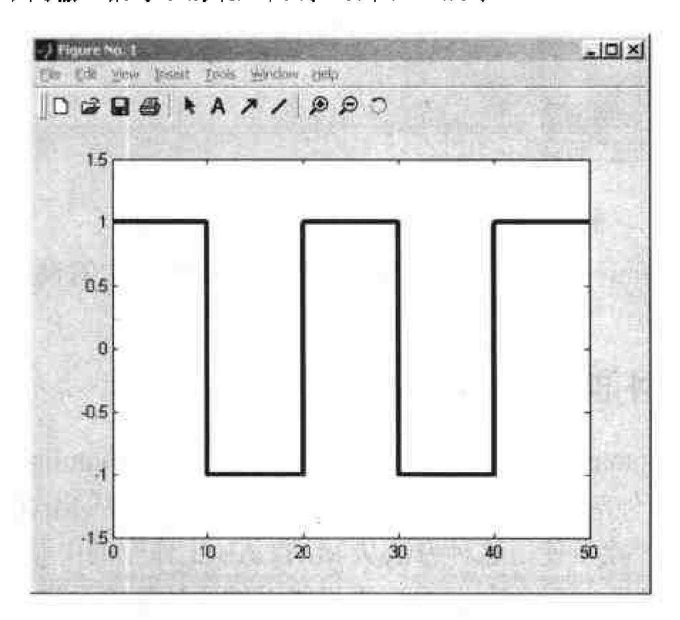


图 8.7　类方波信号

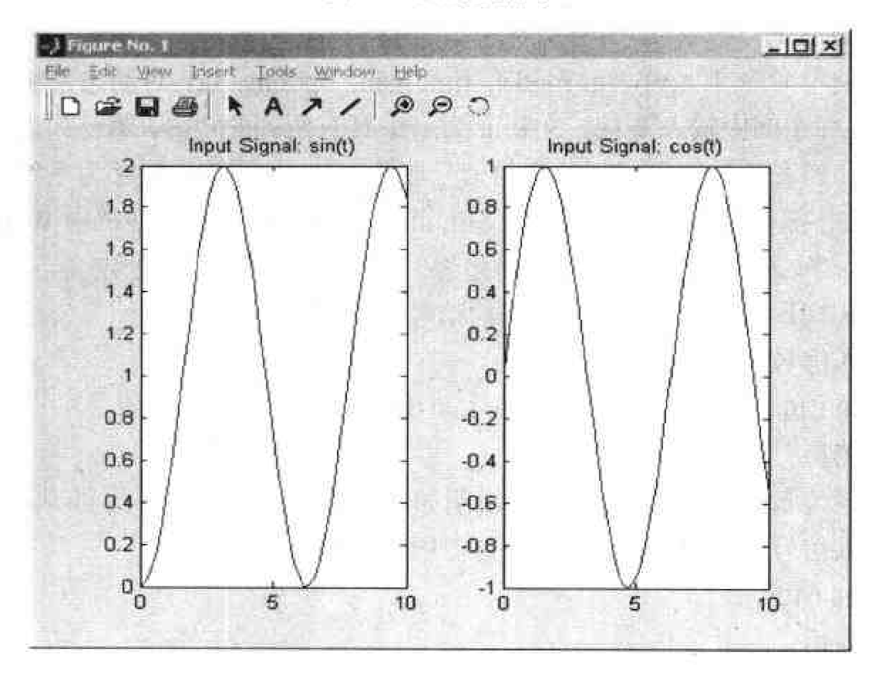


图 8.8　系统在不同输入下的响应曲线

图 8.8 中左侧表示系统输入为 sin(t)时的响应曲线，右侧表示系统输入为 cos(t)时的响应曲线；从系统响应曲线中可以明显看出，当使用 sim 命令的 ut 参数时，Simulink 仿真参数设置对话框中的设置将被覆盖。

8.3.5 simset 与 simget 命令的使用

前面介绍使用 sim 命令进行动态系统仿真时，除了对系统仿真时间与系统输入的设置之外，其它所有的仿真参数均是由 Simulink 的仿真参数设置对话框所设置的。用户也可以使用 simset 对系统仿真参数进行设置，然后再使用命令行方式对系统进行仿真分析，如下所示：

```
>>[t,x,y] = sim(model, timespan, options, ut);
```

其中 options 为仿真选项结构体变量（不包括系统仿真时间），使用 simset 命令可以对此结构体变量进行设置。注意，这里所谓的仿真选项指的就是 Simulink 仿真参数设置对话框 Simulation parameters 中所设置的参数。为了使用户对 options 系统仿真选项有一个总体的了解，首先使用 simget 命令获得表示系统仿真参数的结构体变量，如下所示：

```
>>options=simget('command_in_out')
%获得已经打开的系统模型 command_in_out 的仿真参数选项
options =
          AbsTol : 'auto'
           Debug: 'off'
      Decimation : 1
    DstWorkspace: 'current'
  FinalStateName : ''
       FixedStep : 'auto'
    InitialState : []
     InitialStep : 'auto'
        MaxOrder : 5
      SaveFormat: 'Array'
   MaxDataPoints: 1000
         MaxStep: 'auto'
         MinStep: []
    OutputPoints: 'all'
 OutputVariables: 'ty'
          Refine: 1
          RelTol: 0.0010
          Solver: 'ode45'
    SrcWorkspace: 'base'
           Trace: ''
       ZeroCross: 'on'
```

这里使用 simget 命令所获得的结构体变量包括除仿真时间之外的所有仿真参数选项。

这些仿真参数选项均可以使用 simset 命令进行设置。

1. simset 命令使用介绍

1) 使用语法

```
options = simset(property, value, ...);
options = simset(old_opstruct, property, value, ...);
options = simset(old_opstruct, new_opstruct);
simset
```

2) 功能描述

使用 simset 命令会产生一结构体变量 options，此变量中的各个数据用来设置系统仿真参数。对于没有进行设置的系统仿真参数，Simulink 会使用相应的缺省值。

options=simset(property, value, ...)：设置指定的仿真参数选项值。其中 property 为指定的仿真参数，value 为指定的取值。

options=simset(old_opstruct, property, value, ...)：修改仿真参数结构体变量中已经存在的指定仿真参数选项。其中 old_opstruct 表示已经存在的结构体。

options=simset(old_opstruct, new_opstruct)：合并已经存在的两个仿真参数结构体变量，并且使用 new_opstruct 中的域值覆盖 old_opstruct 中具有相同域名的域值。

simset：显示所有的仿真参数选项及其可能的取值。

3) 仿真参数选项介绍

使用 simset 命令可以设置除仿真时间外的所有系统仿真参数选项，下面对最常用的仿真参数选项及其取值作一个简单的介绍。

(1) AbsTol：取值为一标量，缺省值为 1e–6，表示绝对误差限。仅用于变步长求解器。

(2) Decimation：取值为一正整数，缺省值为 1，表示系统仿真结果返回数据点的间隔（值为 1，表示每一仿真结果数据均返回到相应的变量中；值为 2，表示仿真结果每隔一个数据点返回到相应的变量中，依次类推）。

(3) FixedStep：取值为正值标量，表示定步长求解器的步长。如果对离散系统进行仿真，其缺省取值为离散系统的采样时间；如果对连续系统进行仿真，其缺省取值为仿真时间范围的 1/50。

(4) InitialState：取值为一向量，缺省值为空向量，表示系统的初始状态。如果系统中同时存在连续状态与离散状态，则此向量中先是连续状态的初始值，然后是离散状态的初始值。初始状态的设置会覆盖系统模型中所默认的状态初始值。

(5) InitialStep：取值为一正标量，缺省值为 auto，表示系统仿真时的初始步长（估计值），仅用于变步长求解器；在仿真时求解器首先使用估计的步长，缺省情况下由求解器决定初始仿真步长。

(6) MaxStep：取值为一正值标量，缺省值为 auto，表示系统仿真的最大步长。仅用于变步长求解器，缺省情况下最大仿真步长为仿真时间范围的 1/50。

(7) RelTol：取值为一正值标量，缺省值为 1e–3，表示相对误差限。仅用于变步长求解器。

(8) Solver：表示 Simulink 仿真求解器，其取值为如下由“|”隔开的字符串：VariableStepDiscrete | ode45 | ode23 | ode113 | ode15s | ode23s | FixedStepDiscrete | ode5 | ode4 |

ode3 | ode2 | ode1。在缺省值为变步长连续求解器，运算方法为 ode45（Dormand-Prince）。

(9) ZeroCross：表示仿真过程中的过零检测，取值为 on 或 off，缺省值为 on。仅用于变步长求解器。on 表示对系统的模块进行过零检测，off 表示不进行过零检测。

4) 应用举例

```
>>myoptions=simset('ZeroCross','off');          % 关闭系统仿真过零检测
>>[tout,x,yout]=sim('command_in_out', 10, myoptions);
% 使用 myoptions 仿真参数选项进行系统仿真
```

这里仅介绍了部分最常用的仿真参数选项，至于其它仿真参数选项的意义及其取值就不再介绍了，用户可以参考 simset 命令的详细联机帮助。

2. simget 命令的使用介绍

1) 使用语法

```
struct = simget(model)
value = simget(model, property)
value = simget(OptionStructure, property)
```

2) 功能描述

simget 命令用来获得指定系统模型的仿真参数设置。

struct = simget(model)：获得指定系统模型 model 的所有仿真参数设置结构体变量。

value = simget(model, property)：获得指定系统模型 model 中指定仿真参数 property 的取值。

value = simget(OptionStructure, property)：获得系统仿真长参数选项中指定的仿真参数的取值。其中 property 可以为一单元矩阵（cell matrix）以包含多个系统仿真参数，此时，返回值也为一单元矩阵。

8.3.6　simplot 命令的使用

在观测动态系统仿真结果时，Sinks 模块库中的 Scope 模块是最常用的模块之一。它可以使用户在类似示波器的图形界面中观测系统仿真结果的输出，而非使用诸如 plot 绘制系统的输出。使用 Scope 最重要的优点是可以通过对 Scope 模块的操作，对系统输出进行方便的观测，这一点是 plot 等命令所不及的。在使用命令行方式对动态系统进行仿真时，可以使用 simplot 命令绘制与使用与 Scope 模块相类似的图形。下面介绍 simplot 命令的功能与使用。

1. 使用语法

```
simplot(data);
simplot(time, data);
```

2. 功能描述

simplot 命令用来输出动态系统的仿真结果，且其输出图形与 Scope 模块的输出类似。

3. 参数说明

data：动态系统仿真结果的输出数据（一般由 Outport 模块、To Workspace 模块等产生的输出)。其数据类型可以为矩阵、向量或是结构体等。

time：动态系统仿真结果的输出时间向量。当系统输出数据为带有时间向量的结构体变

量时，此参数被忽略。

4. 举例

仍以前面的 command_in_out 为例说明 simplot 的使用（设系统仿真结果已经存在）。在 MATLAB 命令窗口中键入如下命令：

```
>>simplot(tout2,yout2)          % 使用 simplot 绘制系统仿真结果，如图 8.9 所示
```

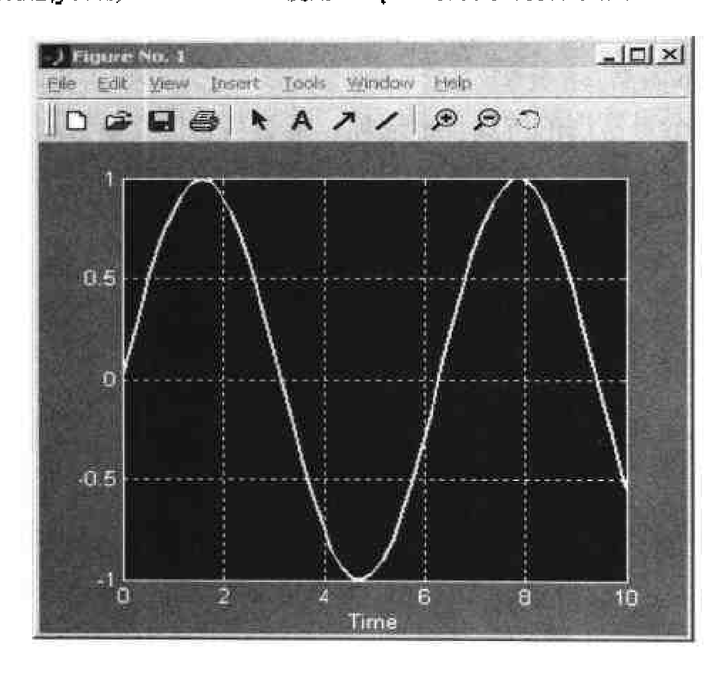


图 8.9 使用 simplot 命令绘制系统仿真结果

8.4 使用 MATLAB 脚本分析动态系统

8.3 节中比较详细地介绍了如何使用命令行方式对动态系统进行仿真，并且以一个简单的系统模型 command_in_out 为例进行说明。掌握这些内容之后，用户便可以使用命令行方式对动态系统进行仿真分析了。然而这并没有发挥命令行方式仿真的最主要的优点：对系统仿真与分析做更多的控制。本节以实际动态系统的仿真分析为例说明如何使用 MATLAB 脚本仿真与分析动态系统，这尤其适合高级用户对设计好的系统进行性能分析。

8.4.1 蹦极跳的安全性分析

在第 5 章中曾给出蹦极跳连续系统模型：

$$m\ddot{x} = mg + bx - a_1\dot{x} - a_2|\dot{x}|\dot{x}$$

其中 m 为蹦极者的质量，g 为重力加速度，x 为物体的位置，$a_1\dot{x}$ 与 $a_2|\dot{x}|\dot{x}$ 表示空气的阻力，而

$$b(x) = \begin{cases} -kx, & x > 0 \\ 0, & x \leqslant 0 \end{cases}$$

表示系在蹦极者身上的弹力绳索对蹦极者位置的影响，这里 k 为弹力绳索的弹性常数。此蹦极跳系统的系统模型框图如图 8.10 所示。

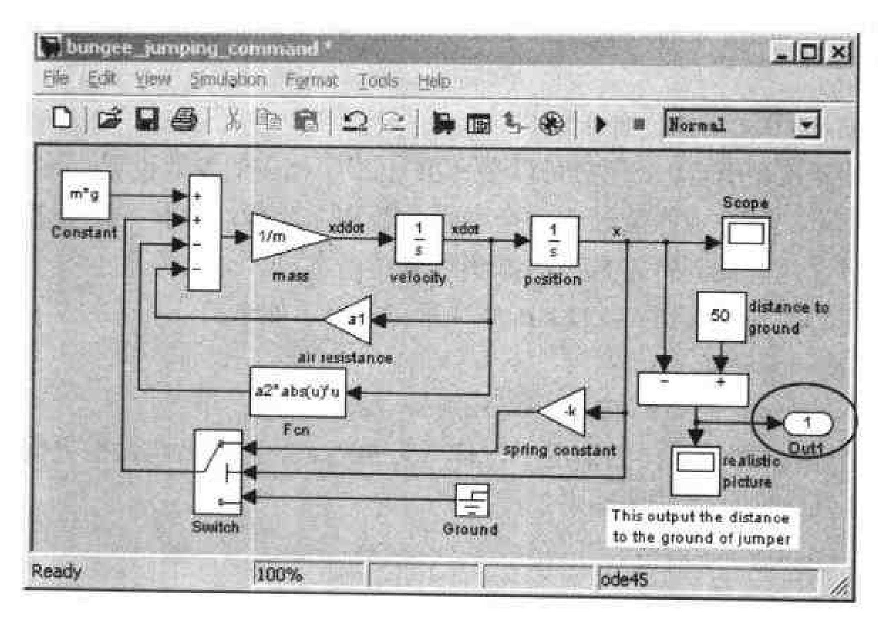


图 8.10　蹦极跳系统模型框图

说明： 系统中所有位置的基准为桥梁所在的位置，即桥梁位置 $x=0$，低于桥梁的位置为正值，高于桥梁的位置为负值。此外，为了使用命令行方式此系统进行分析，利用 Outport 模块将蹦极者的位置输出到 MATLAB 工作空间之中，如图中椭圆框内所示。

在此系统模型中，蹦极者的质量 m=70 kg，重力加速度 g=10 m/s^2，蹦极者的初始位置 $x(0)=-30$，初始速度为 $\dot{x}(0)=0$，桥梁距离地面为 50 m，弹性绳索的弹性常数 k=20；其它的参数 $a_2=a_1=1$。在第 5 章中，按照如上的参数对此系统进行仿真的结果表明：对于体重为 70 kg 的蹦极者，蹦极跳系统的弹力绳索不安全。因为在仿真过程中，蹦极者与地面之间的最小距离小于 0，也就是说蹦极者在此过程中会触地。很显然，如果弹性绳索的弹性常数大于某个值，此蹦极跳系统对于体重为 70 kg 的蹦极者来说才可能是安全的。

下面使用命令行方式对此系统在不同的弹性常数下进行仿真分析，以求出符合安全要求的弹性绳索的最小弹性常数。编写 MATLAB 脚本文件 bungee_jumping_cmd.m，求取最小弹性常数，程序如下所示：

```
m=70;
g=10;
a1=1;                     % 使用 MATLAB 工作空间中变量设置
a2=1;                     % 系统模型中模块的参数
for k=20:50               % 在一定的范围内试验弹性常数
  [t,x,y]=sim('bungee_jumping_command',[0 100]);
   %使用不同的弹性常数进行系统仿真
  if min(y)>0         % 如果仿真结果输出数据的最小值大于 0,
       break;         % 则说明此弹性常数符合安全要求，跳出循环
   end
end
```

```
disp(['The minimum safe k is: ',num2str(k)])        % 显示最小安全弹性常数
dis=min(y);                    % 求取蹦极者与地面之间的最小距离
disp(['The minimum distance between jumper and ground is: ', num2str(dis)])
simplot(t,y)                   % 绘制最小安全弹性常数下系统的仿真结果
```

在运行此 MATLAB 脚本文件求取最小安全弹性常数之前，必须正确设置蹦极者的初始位置 $x(0)$ 与初始速度 $\dot{x}(0)=0$（用户可以参考第 5 章中相应的参数设置，这里不再赘述）。然后再运行此脚本文件，以求取最小安全弹性常数和在此弹性常数下蹦极者与地面之间的最小距离，并且显示在此弹性常数下的蹦极跳系统动态过程。其中最小安全弹性常数、蹦极者与地面之间的最小距离在 MATLAB 命令窗口中显示如下：

```
>> The minimum safe k is: 27
The minimum distance between jumper and ground is: 0.87934
```

在最小安全弹性常数为 27 的情况下，体重为 70 kg 的蹦极者与地面之间的最小距离不足 1 m。图 8.11 所示为此系统的动态过程。

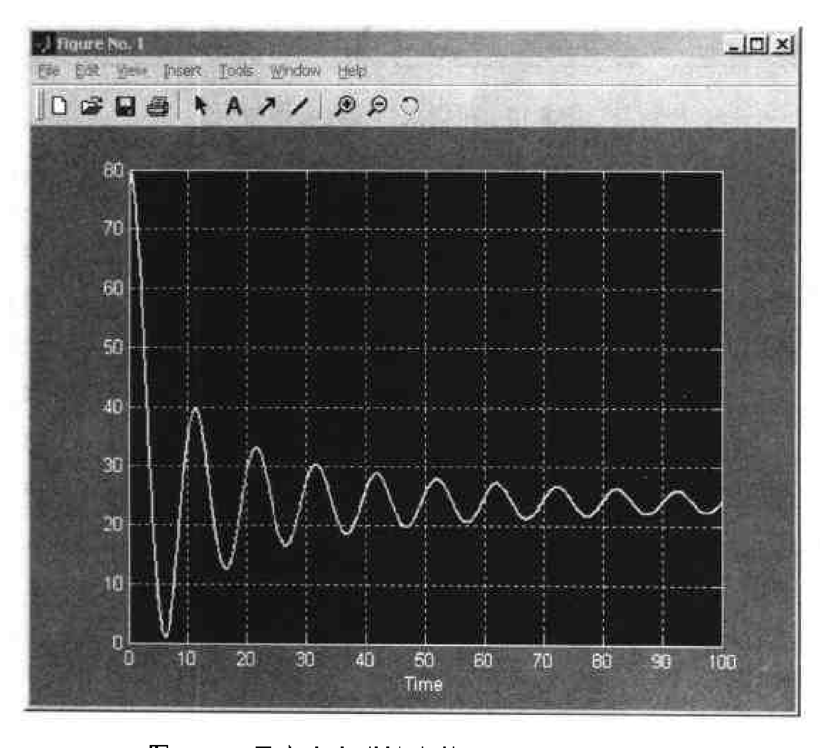


图 8.11　最小安全弹性常数下的系统动态过程

8.4.2　行驶控制系统中控制器的调节

第 5 章中的行驶控制系统给出了系统综合设计仿真的一个简单实例。汽车行驶控制系统的基本目的是控制汽车的速度变化，使汽车的速度在合适的时间之内加速到指定的速度。由第 5 章中的介绍可知，汽车行驶控制系统由如下三部分所构成：

(1) 速度操纵机构的位置变换器。

(2) 行驶控制器。

(3) 汽车动力系统。

其中最为重要的部分为行驶控制器，行驶控制器是一个典型的 PID 反馈控制器。现要求使用命令行方式对行驶控制系统中的行驶控制器中比例调节的性能进行定性的分析。已知行驶控制器中 PID 控制器的 I（积分）、D（微分）参数如下取值分别为 I=0.01、D=1。P（比例）的取值由 MATLAB 脚本文件所决定（以分析不同 P 值下行驶控制器的性能）。为了对行驶控制器中比例调节的性能进行定性的分析，需要对第 5 章的行驶控制系统模型做一些改变，如下所示：

(1) 将行驶控制器子系统中的比例增益的取值改为 p。

(2) 在系统模型的最顶层加入一个 Outport 模块输出仿真结果。

图 8.12 所示为修改后的汽车行驶控制系统的系统模型框图。

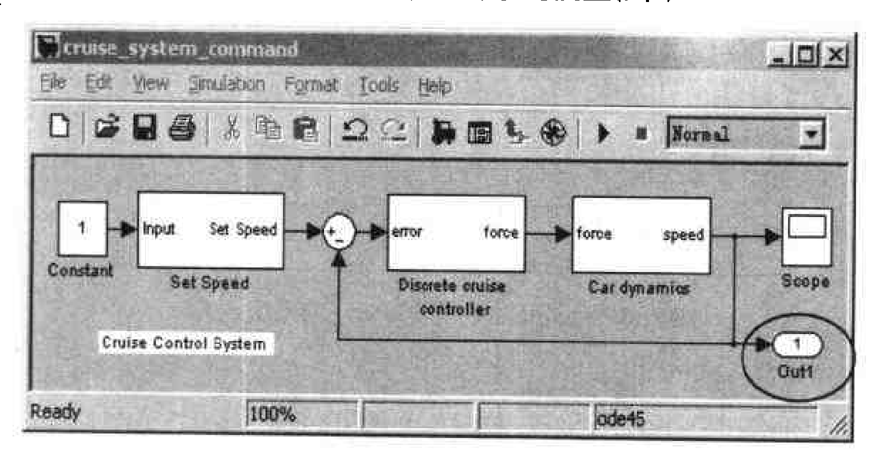


图 8.12　修改后的汽车行驶控制系统模型

然后编写 MATLAB 脚本文件 cruise_system_cmd.m 对行驶控制系统在不同的比例调节器取值下进行仿真，并绘制出不同取值下系统仿真结果以对比例调节性能进行分析。其要求如下：

(1) 编写一个 for 循环改变 p 的值，取值范围为 0 到 25，间隔为 5。

(2) 在同一幅图中作出不同 p 值下系统的仿真结果以对比例调节性能进行分析。编写好的 cruise_system_cmd.m 脚本文件如下所示：

```
for p=0:5:25                                   % 设置不同的比例调节器取值
    [t,x,y]=sim('cruise_system_command');      % 对系统进行仿真
    subplot(3,2,p/5+1)                         % 绘制在此取值下系统仿真结果，其
    plot(t,y)                                  % 结果如图 8.13 所示
    ylabel(['p=',num2str(p)])
end
```

从系统仿真的结果中可以明显看出比例调节器取值对汽车行驶控制系统性能的影响：增加比例调节器的取值可以有效的改善行驶控制系统的动态性能。这是因为，对于行驶控制系统而言，其速度变化越平稳越好（但并非变化缓慢）。从图中可以看出，对于取值较大的比例调节器，汽车速度的过渡时间较小，而且变化平稳（仿真结果曲线无振荡，光滑）。

限于篇幅，本节中仅以此两个动态系统为例说明如何使用 MATLAB 的脚本对动态系统进行更为高级的仿真与分析。相信用户在使用的过程中会不断提高系统设计、仿真与分析的效率，充分体会使用命令行进行系统仿真的强大功能。

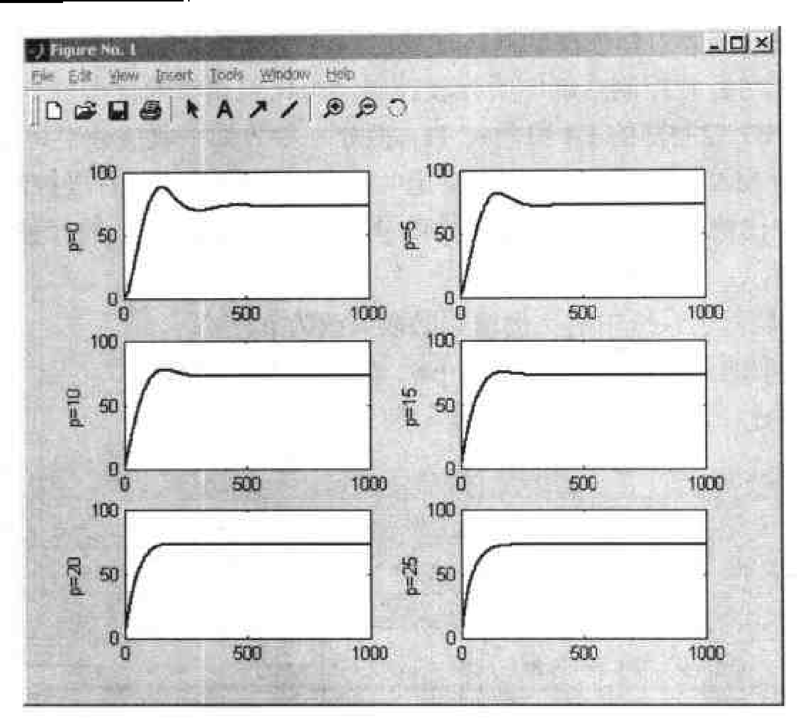


图 8.13　不同比例调节器取值下的系统响应

8.5 其它内容

在对动态系统进行仿真分析时，需要对动态系统的各个方面进行分析以使所设计的系统的动态特性能够满足特殊的要求。在对系统各方面的分析中，确定系统模型中不同模块的状态、系统运行中的平衡点以及对非线性系统进行线性化处理等等是最常见的问题。由于这些技术与使用命令行进行系统仿真分析密切相关，因此在本节中将分别对这三项内容作一个简单而又不失全面的介绍。

8.5.1 系统状态的确定

当使用命令行方式进行动态系统仿真时，经常需要对系统中的状态变量进行分析。当动态系统的系统模型中存在多个状态变量时，对这些状态变量进行分析（尤其是对输出的多个状态变量中的某一个状态进行单独分析），需要清楚地知道状态变量矩阵中各个状态变量的顺序。这时，用户可以使用下面的命令获得系统模型中状态输出的顺序。

```
>> [sizes, x0, xord]=modelname
```

例如，对于蹦极跳系统，如果需要对蹦极者在某个时刻的位置 position 与速度 velocity 进行分析，首先需要设置 Simulink 的仿真参数设置对话框中 Workspace I/O 选项卡以输出系统中的状态变量，如图 8.14 所示。

在设置好系统的仿真参数之后，对系统进行仿真。仿真输出的状态变量矩阵 xout 具有两列数据（即蹦极者距离地面的位置以及当时的速度）。使用如下的命令可以知道动态系统中存在的状态变量以及各个状态在 xout 中的顺序关系：

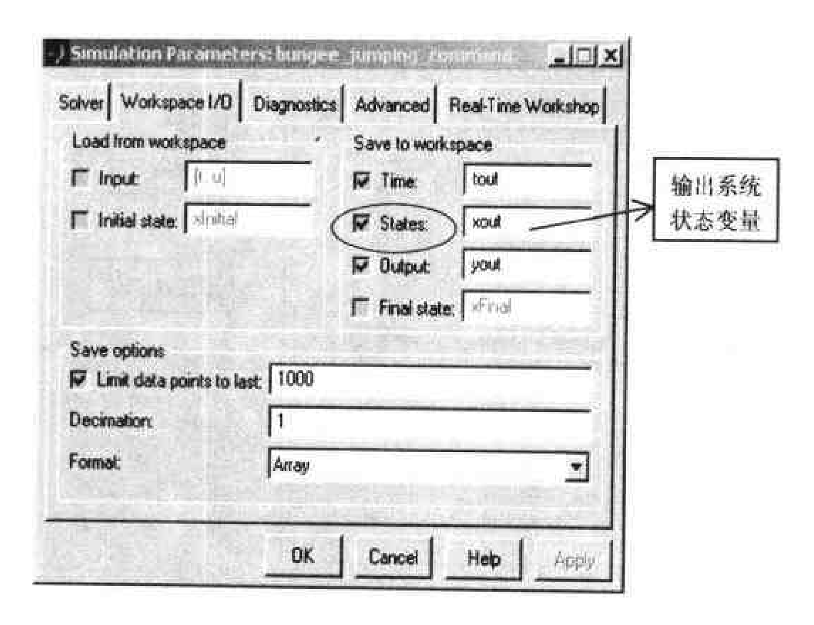


图 8.14　设置 Workspace I/O 选项卡以输出系统状态

```
>> [sizes, x0,xord]=bungee_jumping_command
sizes =                                   % 尺寸向量
     2                                    % 表示连续状态数目
     0                                    % 表示离散状态数目
     1                                    % 表示仿真结果输出数目
     0                                    % 表示系统外部输入数目
     0                                    % 系统保留
     0                                    % 直接馈通标记
     0                                    % 表示系统采样时间
x0 =                                      % 状态变量初始值
  -30
    0
xord =                                    % 系统中的状态变量输出顺序
    'bungee_jumping_command/position'
    'bungee_jumping_command/velocity'
```

从中可以知道蹦极跳系统的诸多信息，如系统的输入输出变量的数目（外部输入数目为 0、输出 1 个变量）、系统中产生状态变量的模块及其输出顺序（第一个状态为积分块 position 的输出，即蹦极者的位置；第二个状态为积分块 velocity 的输出，即蹦极者的速度）、各状态变量的初始取值（积分块 position 的初始取值为-30，积分块 velocity 的初始取值为 0）等等信息。此时使用 MATLAB 的 Array Editor 观测输出变量 xout，其中第一列与第二列分别为蹦极者的位置与速度，如图 8.15 所示。

在准确知道系统仿真结果输出状态的相互顺序关系后，便可以对其进行进一步的分析，这里不再赘述。

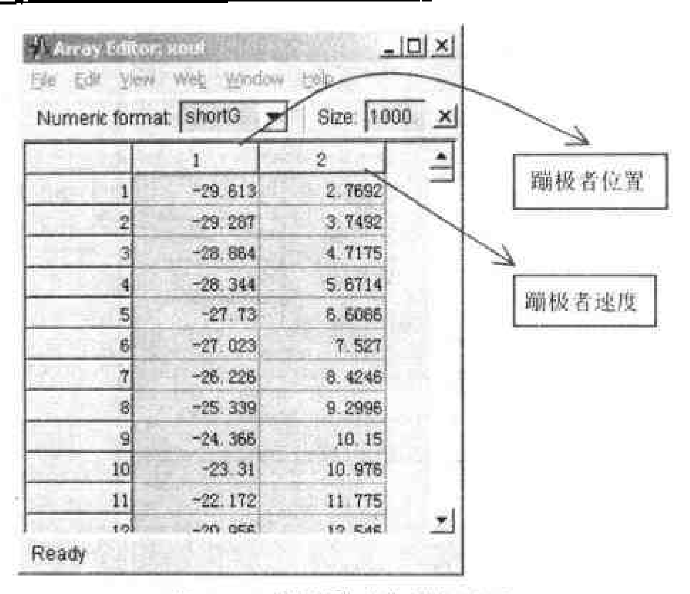


图 8.15 蹦极跳者位置与速度

8.5.2 系统平衡点的确定

在大多数的系统设计中，设计者都需要对所设计的系统进行稳定性分析，因为绝大多数的系统在运行之中，都需要按照某种方式收敛到指定的平衡点处。这里所谓的平衡点一般指的是系统的稳定工作点，此时系统中所有的状态变量的导数均为 0，系统处于稳定的工作状态。本书称处于平衡点状态时的系统为静止系统。

在系统设计中，最主要的设计目的之一便是使系统能够满足系统稳定的要求并在指定的平衡点处正常工作。在使用 Simulink 进行动态系统设计、仿真与分析时，可以使用命令 trim 对系统的稳定性与平衡点进行分析。下面简单介绍一下 trim 的使用与功能。

1. 使用语法

```
[x,u,y,dx] = trim('sys')
[x,u,y,dx] = trim('sys',x0,u0,y0)
[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy)
[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx)
[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx,options)
[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx,options,t)
[x,u,y,dx,options] = trim('sys',...)
```

2. 功能描述

根据系统的输入、初始状态（也可以说是初始的工作点），按照一定的方法求取系统中距离此工作点最近的平衡点，以及在达到平衡点时的系统输入与输出。如果系统中不存在平衡点，则 trim 命令会返回系统状态变量最接近 0 的工作点。

[x,u,y] = trim('sys')：求取距离给定初始状态 x0 最近的系统平衡点。

[x,u,y] = trim('sys',x0,u0,y0)：求取距离给定初始状态 x0、初始输入 u0 与初始输出 y0 最近的平衡点。

[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy)：求取距离给定初始值向量中某一初始值距离最近的平衡点。其中 x0、u0 与 y0 为初始值向量，ix、iu 与 iy 为相应的初始值序号。

[x,u,y,dx] = trim('sys', x0, u0, y0, ix, iu, iy, dx0, idx)：求取非平衡点，此平衡点处的系统状态为指定值。其中 dx0 为指定状态值向量，idx 为相应的序号。

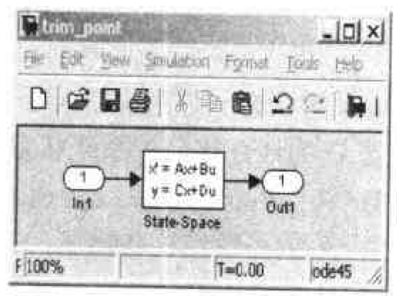
至于 trim 命令中的 options 选项，其功能是用来优化平衡点的求取，这里不再赘述。需要使用此功能的用户可以参考 Simulink 中 trim 命令的联机帮助。

3. 注意事项

在使用 trim 命令求取系统的平衡点时，所求取的平衡点为局部最优平衡点，而非全局最优平衡点，因此如果需要求取全局最优平衡点，需要使用多个初始状态进行搜索。这是因为对于很多系统而言，系统中平衡点的数目往往不止一个。

4. 应用举例

对于图 8.16 所示的系统，试求取其在初始状态 x0=[1;1]以及初始输入 u0=[1;1]下的平衡点。



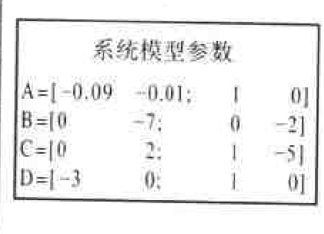


图 8.16　由状态空间描述的线性系统模型框图

其实只需要在 MATLAB 的命令窗口中键入如下的命令便可以求取系统在此初始状态与初始输入下的平衡点：

```
>> x0=[1;1];
>>u0=[1;1];
>>[x,u,y] = trim('trim_point', x0, u0);
```

则在 MATLAB 命令窗口中显示如下结果：

```
x =                              % 系统所处平衡点处的状态变量值
    1.0e-012 *
    0.1716
    0.0840
u =                              % 系统在平衡点处的输入值
    0.3333
    0.0000
y =                              % 系统在平衡点处的输出值
    -1.0000
    0.3333
```

8.5.3　非线性系统的线性化处理

迄今为止，线性系统的设计与分析技术已经非常完善了。但在实际的系统中，很少有真正的线性系统，大部分的系统都是非线性系统，然而对非线性系统的设计与分析还处于

发展的初期，其设计与分析还主要依赖于设计者的设计经验。这给系统设计、仿真与分析带来了很多的不便。在系统设计中，往往将非线性系统简化为线性系统，这是因为在一定的范围之内，非线性系统的动态性能可以用线性系统来代替。Simulink 提供的 linmod、dlinmod 命令便可以在某一操作点（operating point）处对非线性系统进行线性化处理，Simulink 通过在操作点附近进行微小的扰动的方法对模型线性化。下面对 linmod 与 dlinmod 进行介绍。

1. 使用语法

```
[A,B,C,D] = linmod('sys', x, u)
[num,den] = linmod('sys', x, u)
sys_struc = linmod('sys', x, u)
[Ad,Bd,Cd,Dd] = dlinmod('sys', Ts, x, u)
[numd,dend] = dlinmod('sys', Ts, x, u)
```

2. 功能描述

[A,B,C,D] = linmod('sys', x, u)：在指定的系统状态 x 与系统输入 u 下对系统 sys 进行线性化处理，x 与 u 的缺省值为 0。A、B、C 与 D 为线性化后的系统状态空间描述矩阵。

[num,den] = linmod('sys', x, u)：num 与 den 为线性化后的系统传递函数描述。

sys_struc = linmod('sys', x, u)：返回线性化后的系统结构体描述，其中包括系统状态名称、输入与输出名称以及操作点的信息。

[Ad,Bd,Cd,Dd] = dlinmod('sys', Ts, x, u)：可以对非线性、多速率混合系统（包括离散系统与连续系统）进行线性化处理。其中 Ts 为系统的采样时间，Ts=0 表示将离散系统线性化为连续系统。返回线性化后系统的状态控制描述。

[numd,dend] = dlinmod('sys', Ts, x, u)：返回线性化后系统的传递函数描述。

3. 应用举例

这里仍以蹦极跳系统为例说明非线性系统的线性化处理，从蹦极跳系统的系统动力学方程可知，蹦极跳系统为一非线性系统。下面对其进行线性化处理（其中系统模型中参数符合安全要求，即弹性常数为 27），操作如下所示：

```
>> [A,B,C,D]=linmod('bungee_jumping_command')
% 返回线性化后系统的状态空间描述
A =
     0    1.0000
     0   -0.0143
B =
   Empty matrix: 2-by-0
C =
   -1.0000         0
D =
   Empty matrix: 1-by-0
```

由于系统没有输入信号，所有矩阵 B、D 均为空矩阵。又如：

```
>> sys_struction=linmod('bungee_jumping_command')
```

```
% 返回线性化后系统的结构体描述
sys_struction =
             a: [2x2 double]
             b: [2x0 double]
             c: [-1.0000 0]
             d: [1x0 double]
     StateName: {2x1 cell}
    OutputName: {'bungee_jumping/Out1'}
     InputName: {0x1 cell}
     OperPoint: [1x1 struct]
            Ts: 0
```

此结构体内容包括了线性化后系统的如下信息：系统的状态空间描述矩阵 a、b、c 及 d，系统模型中的状态变量名称 StateName，系统输出结果名称 OutputName，系统输入信号名称 InputName，系统的操作点 OperPoint 以及系统采样时间 Ts 等。用户可以从此结构体变量中获得需要的信息，如

```
>> statename=sys_struction.StateName
%获得系统模型中状态变量名称
statename =
    'bungee_jumping/position'
    'bungee_jumping/velocity'
```

此外，dlinmod 命令的使用方法与 linmod 完全一致。使用这些命令可以非常容易地对比较复杂的动态系统进行简化，在某个范围内使用较为简单的线性系统取代复杂的非线性系统，从而可以大大简化系统的实现。至于如何简化系统的实现，则超出了本书的范围，感兴趣的用户可以参考有关系统实现的书籍。

8.6 回调函数

8.4 节中介绍使用命令行方式对动态系统进行仿真时，以蹦极跳系统为例对其进行了说明，并编写了相应的 MATLAB 脚本文件 bungee_jumping_cmd.m 对此系统进行了分析，以求出满足安全性要求的最小安全弹性常数。在 bungee_jumping_cmd.m 文件中使用普通的 MATLAB 命令对系统模型中的某些模块参数进行了设置（即对相应的变量进行正确的赋值）。除此之外，还可以使用 Simulink 的回调函数对系统模型中的参数进行相应的操作，或完成其它的任务。

1. 回调函数的基本概念

所谓回调函数，一般是指系统模型或系统模型中的某些模块在特定的时刻所运行的一系列用户自定义的命令的集合。对于系统模型与其中的系统模块而言，它们拥有不同的回调函数。这里特定的时刻一般是指系统中发生特定事件的时刻，如打开系统模型、对系统模型进行仿真、对模型中的模块进行操作以及打开模块等等。用户可以使用 set_param 命令

设置系统模型或其中模块的回调函数，或使用 get_param 命令获得相应的回调函数的设置，至于 set_param、get_param 的使用在 8.1 节中已经做了介绍，这里不再赘述。

下面仍以蹦极跳系统为例说明回调函数的设置与使用，最后简单介绍一下 Simulink 中系统模型与系统模块所支持的回调函数。

2. 回调函数的设置与使用举例

使用 MATLAB 脚本命令对蹦极跳系统进行仿真分析，要求使用回调函数设置系统模型中的参数，并在仿真结束后绘制系统的相平面图（系统中两个状态之间的关系图，即蹦极者速度相对于其位置的关系图）。为了绘制系统的相平面图，需要输出系统中的状态变量，因此可以使用 Simulink 的参数设置对话框中的 Workspace I/O 选项设置输出状态（选中 States 复选框即可）。然后依次在 MATLAB 命令窗口中键入如下的命令：

```
>>set_param('bungee_jumping_command','InitFcn','m=70;g=10;a1=1;a2=1;k=27;');
    % 'InitFcn'表示在系统仿真开始的时候开始调用，以设置模型参数
>>set_param('bungee_jumping_command','StopFcn', 'plot(xout(:,1),xout(:,2))');
    % 'StopFcn'表示在系统仿真结束时开始调用，以绘制相平面图
>>sim('bungee_jumping_command',[0 100]);
    % 完成系统仿真
```

此时所绘制的相平面图如图 8.17 所示，其中横坐标表示蹦极者相对于桥梁的位置，纵坐标表示蹦极者在某一位置的速度。

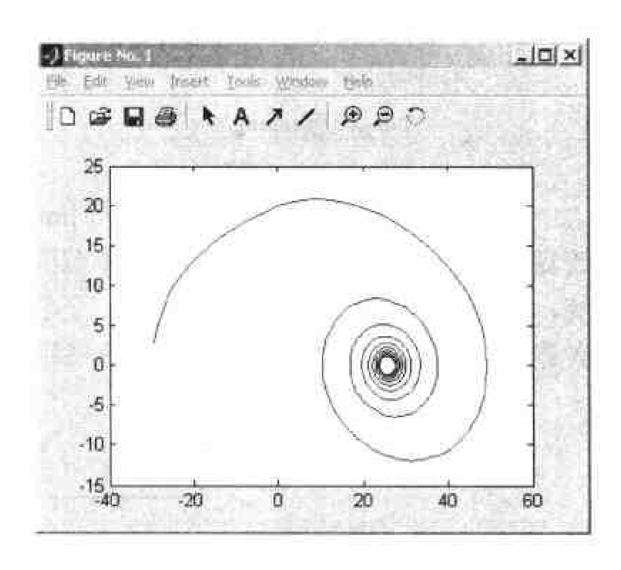


图 8.17　蹦极跳系统的相平面图

3. 其它回调函数简介

1) 模型回调函数

CloseFcn：在系统模型框图关闭之前执行。

PostLoadFcn：在系统模型加载完成后执行。当编写一个要求模型完全加载后方能启动的界面程序时非常有用。

InitFcn：在系统模型仿真开始的时候调用。使用此回调函数可以在系统仿真开始时对系统模型中的参数进行设置。

PostSaveFcn：在系统模型保存后执行。

PreLoadFcn：在系统模型加载之前执行。

PreSaveFcn：在系统模型保存之前执行。

StartFcn：在系统仿真开始之前执行。

StopFcn：在系统仿真结束后执行。注意，在 StopFcn 执行之前系统仿真结果已经输出到 MATLAB 工作空间或是数据文件中。

2) 系统模块回调函数

CloseFcn：在使用 close_system 命令关闭系统模块时执行。

CopyFcn：在一个系统模块被复制后执行。此函数对于子系统中的模块是递归调用的。

DeleteFcn：在系统模块被删除之前执行。此函数对于子系统中的模块也是递归调用的。

DestroyFcn：在系统模块被清除后执行。

InitFcn：在系统框图被编译以及系统模块参数被求值之前执行。

LoadFcn：在系统框图加载后执行。此函数对于子系统中的模块是递归调用的。

ModelCloseFcn：在系统框图关闭前执行。此函数对于子系统中的模块是递归调用的。

MoveFcn：在系统模块移动或改变尺寸时执行。

NameChangeFcn：在系统模块的名称（或是路径）改变时执行。此函数对于子系统模块中的模块是递归调用的。

OpenFcn：在系统模块打开时运行。此函数通常与子系统模块结合使用。例如，在双击一个子系统模块或将其作为参数来调用 open_system 命令时运行。OpenFcn 回调函数可以改变通常模块打开时的行为方式（即打开模块参数设置对话框）。

ParentCloseFcn：在关闭包含此系统模块的子系统之前（或作为通过 new_system 命令建立的新的子系统中的一部分时）执行。

PreSaveFcn：在系统框图保存前执行。对子系统中的模块递归调用执行。

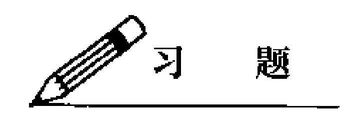
PostSaveFcn：在系统框图保存后执行。对子系统中的模块递归调用执行。

StartFcn：在系统框图被编译之后，系统仿真开始之前执行。对于 S-函数模块，StartFcn 恰好在第一次执行模块的 mdlProcessParameterS-函数前执行。

StopFcn：在任何系统仿真终止的时候执行。对于 S-函数块，StopFcn 在模块的 mdlTerminate 函数执行之后执行。

UndoDeleteFcn：在被删除的系统模块被恢复后执行。

合理地使用 Simulink 系统模型与系统模块的回调函数，可以完成许多单独使用命令行所不易完成的任务，从而进一步提高系统设计与仿真的效率。希望用户能够在使用的过程中仔细体会命令行仿真方式的使用，进而熟练掌握命令行仿真技术以更好地对系统进行设计与仿真分析。



习　题

如图 8.18 所示的比例微分控制系统，使用命令行方式对系统进行仿真分析。要求如下：

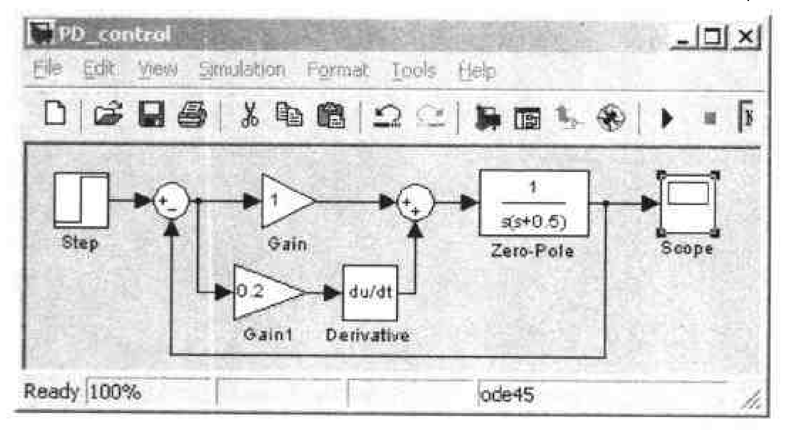


图 8.18　比例微分控制系统模型框图

(1) 系统输入为单位阶跃信号，阶跃时刻为 0（即在时刻 0 处信号取值由 0 变为 1）。

(2) 分析在比例系数 P = 1，微分系数 D 分别为 0.1、0.2、0.3 及 0.4 时控制系统的响应；并绘制系统响应曲线进行比较，分析微分控制的特点。

(3) 分析在微分系数 D = 0.2，比例系数 P 分别为 0.5、1.0、1.5 及 2.0 时控制系统的响应；并绘制系统响应曲线进行比较，分析比例控制的特点。

提示：(1) 修改系统模型，将系统仿真结果输出至 MATLAB 工作空间中。

(2) 设置系统仿真参数对话框中的 Workspace I/O，从 MATLAB 工作空间中输入微分系数 D 与比例系数 P。

(3) 编写 M 文件，使用 for 循环与 sim 命令完成系统仿真与分析。(参考 8.4.2 节中行驶控制系统中控制器的调节。)

第9章 S-函数

内容概要

- S-函数概述
- S-函数的工作原理
- 编写 M 文件 S-函数
- 编写 C MEX S-函数

S-函数无疑是 Simulink 最具魅力的地方，它完美地结合了 Simulink 框图简洁明快的特点和编程灵活方便的优点。它提供了增强和扩展 Simulink 能力的强大机制，同时也是使用 RTW（Real Time Workshop）实现实时仿真的关键。实际上 Simulink 许多模块所包含的算法均是用 S-函数写的，用户也可以编写自己的 S-函数，然后进行封装便可得到具有特定功能的定制模块。S-函数支持 MATLAB、C、C++、FORTRAM 以及 Ada 等语言，使用这些语言，按照一定的规则就可以写出功能强大的模块。本章将介绍 S-函数的基本概念、工作原理以及如何使用和编写 S-函数。

9.1 S-函数概述

9.1.1 S-函数的基本概念

S-函数是系统函数(System Function) 的简称，是指采用非图形化的方式(即计算机语言，区别于 Simulink 的系统模块）描述的一个功能块。用户可以采用 MATLAB 代码，C，C++，FORTRAM 或 Ada 等语言编写 S-函数。S-函数由一种特定的语法构成，用来描述并实现连续系统、离散系统以及复合系统等动态系统：S-函数能够接收来自 Simulink 求解器的相关信息，并对求解器发出的命令做出适当的响应，这种交互作用非常类似于 Simulink 系统模块与求解器的交互作用。一个结构体系完整的 S-函数包含了描述动态系统所需的全部能力，所有其他的使用情况都是这个结构体系的特例。往往 S-函数模块是整个 Simulink 动态系统的核心。

S-函数作为与其他语言相结合的接口，可以使用这个语言所提供的强大能力。例如，MATLAB 语言编写的 S-函数可以充分利用 MATLAB 所提供的丰富资源，方便地调用各种工具箱函数和图形函数；使用 C 语言编写的 S-函数则可以实现对操作系统的访问，如实现与其它进程的通信和同步等。

用户可能会有如下的疑问：Simulink 已经提供了大量内置的系统模块，并且允许用户自定义模块，那么为何还要使用 S-函数呢？诚然，对于大多数动态系统仿真分析而言，使用 Simulink 提供的模块即可实现，而无需使用 S-函数。但是，当需要开发一个新的通用的模

块作为一个独立的功能单元时，使用 S-函数实现则是一种相当简便的方法。另外，由于 S-函数可以使用多种语言编写，因此可以将已有的代码结合进来，而不需要在 Simulink 中重新实现算法，从而在某种程度上实现了代码移植。此外，在 S-函数中使用文本方式输入公式、方程，非常适合复杂动态系统的数学描述，并且在仿真过程中可以对仿真进行更精确的控制。

简单来说，用户可以从如下的几个角度来理解 S-函数：

(1) S-函数为 Simulink 的“系统”函数。

(2) 能够响应 Simulink 求解器命令的函数。

(3) 采用非图形化的方法实现一个动态系统。

(4) 可以开发新的 Simulink 模块。

(5) 可以与已有的代码相结合进行仿真。

(6) 采用文本方式输入复杂的系统方程。

(7) 扩展 Simulink 功能。M 文件 S-函数可以扩展图形能力，C MEX S-函数可以提供与操作系统的接口。

(8) S-函数的语法结构是为实现一个动态系统而设计的（默认用法），其它 S-函数的用法是默认用法的特例（如用于显示目的）。

9.1.2 如何使用 S-函数

前面简单介绍了 S-函数的基本概念及其主要的功能。在动态系统设计、仿真与分析中，用户可以使用 Functions & Tables 模块库中的 S-function 模块来使用 S-函数；S-function 模块是一个单输入单输出的系统模块，如果有多个输入与多个输出信号，可以使用 Mux 模块与 Demux 模块对信号进行组合和分离操作。当然，S-function 模块仅仅是以图形的方式提供给用户一个 S-函数的使用接口，实际的功能是由 S-函数源文件来描述的。在 S-function 模块的参数设置对话框中仅包括 S-函数的名称与函数的参数列表（如图 9.1 所示），而 S-函数源文件必须由用户自行编写。

一般而言，S-函数的使用步骤如下：

(1) 创建 S-函数源文件。创建 S-函数源文件有多种方法，当然用户可以按照 S-函数的语法格式自行书写每一行代码，但是这样做容易出错且麻烦。Simulink 为我们提供了很多 S-函数模板和例子，用户可以根据自己的需要修改相应的模板或例子即可。

(2) 在动态系统的 Simulink 模型框图中添加 S-function 模块，并进行正确的设置。

(3) 在 Simulink 模型框图中按照定义好的功能连接输入输出端口。

注意：S-function 模块中 S-函数名称、S-函数参数列表必须和用户建立的 S-函数源文件的名称、函数参数完全一致（包括参数的顺序）；并且函数参数之间必须使用逗号隔开，如图 9.1 所示。此外，用户也可以使用子系统技术对 S-函数进行封装，以增强系统模型的可读性。

为了方便 S-函数的使用和编写，Simulink 的 Functions & Tables 模块库还提供了 S-function builder 模块与 S-function demos 模块组。其中 S-function builder 为用户编写 C MEX S-函数提供了一个图形化的集成开发环境，用户只需在相应的位置填入相应的名称和代码即可编译成相应的 Mex 文件；而 S-function demos 模块为用户提供了编写 S-函数的

各种例子，以及 S-函数模板模块。

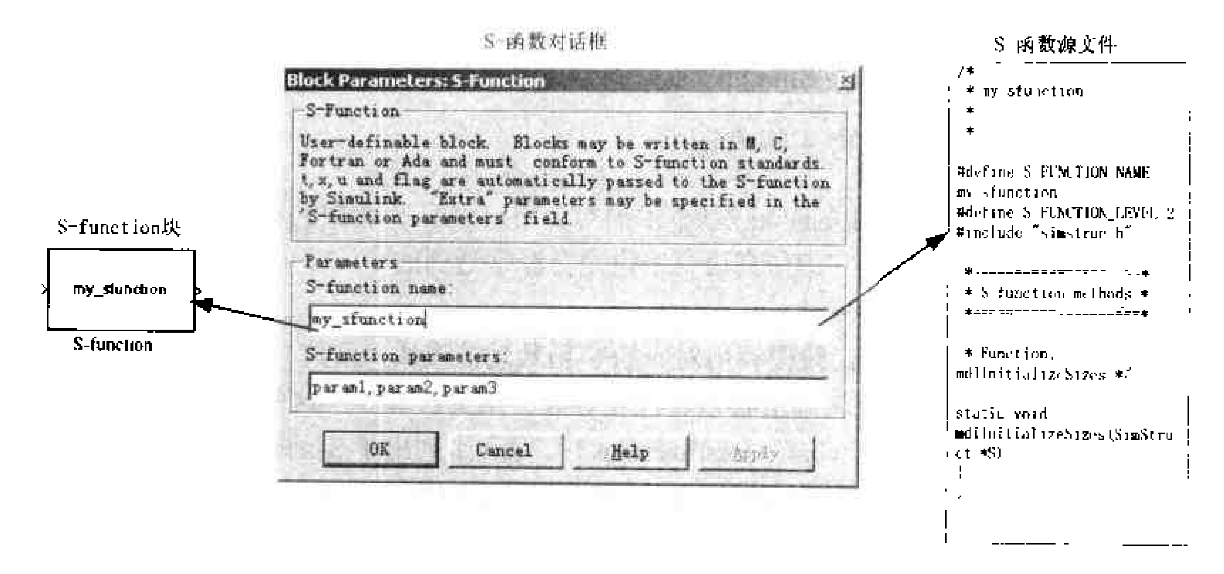


图 9.1　S-函数块对话框

让我们先看一个最简单的 S-函数，它实现了输入乘以 2 然后输出。

【例 9.1】　使用 S-函数实现系统：y=2u。

解：(1) 打开模板 M 文件 S-函数模板文件 sfuntmpl.m ，在\MATLABroot\work 目录下另存为 doublesfunction.m。

(2) 找到函数 mdlInitializeSizes，修改以下代码：

```
sizes.NumOutputs = 1;
sizes.NumInputs = 1;
```

(3) 找到函数 mdlOutputs，加入以下代码：

```
sys=2*u;
```

(到现在为止我们的第一个 S-函数写完了。下面演示一下它的作用。)

(4) 在 Simulink 空白页中添加 S-function 块，打开 S-function 块对话框，参数 S-function name 设置为 doublesfunction 。按照图 9.2 添加连接好其余的各个模块。

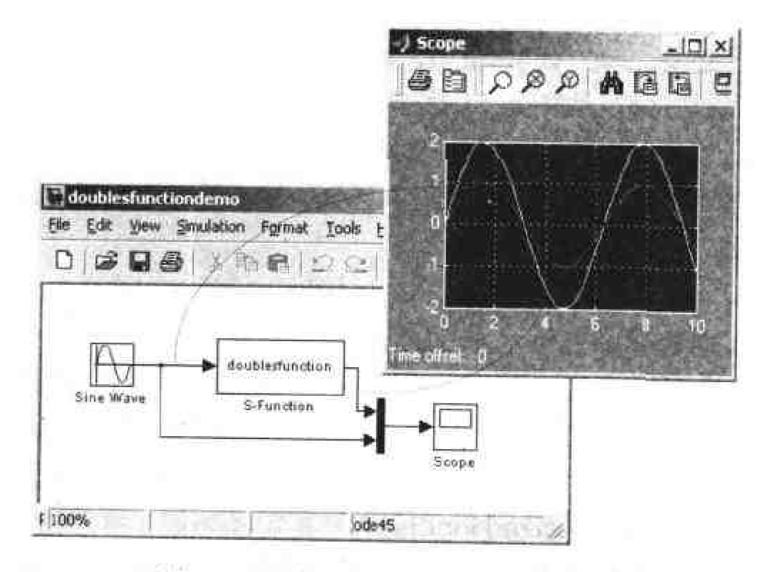


图 9.2　一个用 S-函数实现的简单系统

(5) 开始仿真，在 Scope 中观察输出结果，可以看到输入正弦信号被放大为原来的 2 倍，如图 9.2 所示。

注意：不要把 S-函数和 Simulink 框图命名为相同的名字。

9.1.3 与 S-函数相关的一些术语

理解下列与 S-函数相关的一些基本术语对于用户理解 S-函数的概念与编写都是非常有益的；而且这些概念在其它的仿真语言中也是会经常遇到的。

1. 仿真例程（Routines）

Simulink 在仿真的特定阶段调用对应的 S-函数功能模块（函数），来完成不同的任务，如初始化、计算输出、更新离散状态、计算导数、结束仿真等，这些功能模块（函数）称为仿真例程或者回调函数（call back functions）。表 9.1 列出了 S-函数例程函数和对应的仿真阶段。关于仿真例程将在 S-函数工作原理一节详细介绍。

表 9.1 S-函数例程

S-函数仿真例程	仿真阶段
mdlInitialization	初始化
mdlGetTimeofNextVarHit	计算下一个采样点
mdlOutput	计算输出
mdlUpdate	更新离散状态
mdlDerivatives	计算导数
mdlTerminate	结束仿真

2. 直接馈通（Direct feedthrough）

直接馈通意味着输出或可变采样时间与输入直接相关（详见第 6 章）。在如下的两种情况下需要直接馈通：

(1) 某一时刻的系统输出 y 中包含某一时刻的系统输入 u。

(2) 系统是一个变采样时间系统（variable sample time system）且采样时间计算与输入 u 相关。

正确设置馈通标志（feedthrough flag）是非常重要的，因为这不仅关系到系统模型中系统模块的执行顺序，还关系到对代数环的检测与处理（用户可以参考第 6 章中有关代数环的介绍）。

3. 采样时间和偏移量（Sample time & offsets）

采样时间在离散时间系统内控制采样时间间隔，偏移量则用于延迟采样时间点（sample time hits）。它们有如下关系：

$$time = (n \times sample_time_value) + offset_time$$

其中 n 表示第 n 个采样点。

Simulink 在每一个采样点上调用 mdlOutput 和 mdlUpdate 例程。对于连续时间系统采样时间和偏移量的值应该设置为零。采样时间还可以继承自驱动模块、目标模块或者系统最小采样时间，这种情况下采样时间值应该设置为-1，或者 INHERITED_SAMPLE_TIME。

4. 动态输入（Dynamically sized inputs）

S-函数支持动态可变维数的输入。S-函数的输入变量 u 的维数决定于驱动 S-函数模块的

输入信号的维数。所以当仿真开始的时候，需要先估计 S-函数的输入维数。在 M 文件 S-函数中动态设置输入维数时，应该把 sizes 数据结构的对应成员设置为-1 或者 DYNAMICLLY_SIZED。在 C 文件 S-函数中需要调用函数 ssSetInputPortWidth 来动态设置输入维数。其它的如状态维数和输出维数同样是动态可变的。

9.2　S-函数的工作原理

了解 S-函数的工作原理对于用户编写 S-函数无疑是非常有帮助的，对于理解 Simulink 的整个仿真原理也是很有益的。本节先简单地介绍作为 S-函数数学基础的状态方程，然后介绍 Simulink 仿真过程和 S-函数的仿真流程。

9.2.1　状态方程

在对动态系统建模时，总是能够采用广义的状态空间形式对无论是线性系统还是非线性系统进行描述。这个描述包含以下两个方程：

状态方程：$\dot{x} = f_c(x,u,t)$

输出方程：$y = g(x,u,t)$

状态方程描述了状态变量的一阶导数与状态变量、输入量之间的关系。n 阶系统具有 n 个独立的状态变量，系统状态方程则是 n 个联立的一阶微分方程或者差分方程。对于一个系统，由于所选择的状态变量不同，会导出不同的状态方程，因此状态方程的形式不是唯一的。输出方程描述了输出与状态变量、输入量之间的关系。输出量根据任务的需要确定。一个典型的线性系统的状态方程可以用矩阵的形式描述为：

状态方程：$\dot{x} = Ax + Bu$

输出方程：$y = Cx + Du$

其中 A、B、C、D 分别是状态矩阵、输入矩阵、输出矩阵、前馈矩阵。

Simulink 框图的大部分模块都具有一个输入向量 u、一个输出向量 y 和一个状态向量 x，如图 9.3 所示。

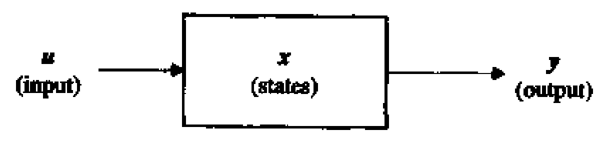


图 9.3　Simulink 模块

u，x，y 和时间 t 之间存在如下关系：

输出方程：$y = f_o(t,x,u)$

连续状态方程：$dx = f_d(t,x,u)$

离散状态方程：$x_{k+1} = f_u(t,x,u)$

其中 $x = [dx \quad x_{k+1}]$。

这实际上正是一个状态方程，描述了输入 y 和输出 u 之间的微分或差分关系。状态向

量 x 分为两部分：连续时间状态 dx 和离散时间状态 x_{k+1}，连续状态占据了状态向量的第一部分，离散状态占据了状态向量的第二部分。

S-函数同样是一个 Simulink 模块。它的以下几个例程函数清楚地体现了状态空间所描述的特性。

(1) S-函数中的连续状态方程描述。状态向量的一阶导数是状态 x、输入 u 和时间 t 的函数。在 S-函数中，状态的一阶导数是在 mdlDerivatives 例程中计算的，并将结果返回供求解器积分。

(2) S-函数中的离散状态方程描述。下一步状态 x_{k+1} 的值依赖于当前的状态 x_k 输入 u 和时间 t。这是通过 mdlUpdate 例程完成的，并将结果返回供求解器在下一步时使用。

(3) S-函数中的输出方程描述，输出值是状态、输入和时间的函数。这个方程不应当包含任何动态方程（微分或差分）在内。输出值是在 mdlOutputs 中计算的，并通过求解器传递给其它模块。

9.2.2 Simulink 仿真的两个阶段

理解 S-函数首先要很好地了解 Simulink 的仿真过程。仿真包含两个主要阶段，第一个阶段是初始化，这时块的所有参数都已确定下来。初始化阶段完成了以下工作：

(1) 传递参数给 MATLAB 进行求值。

(2) 得到的数值作为实际的参数使用。

(3) 展开模型的层次，每个子系统被它们所包含的块替代。

(4) 检查信号的宽度和连接。

(5) 确定状态初值和采样时间。

初始化过后，进入仿真的第二个阶段——运行阶段，仿真开始运行。仿真过程是由求解器和系统（Simulink 引擎）交互控制的。求解器的作用是传递块的输出，对状态导数进行积分，并确定采样时间。系统的作用是计算块的输出，对状态进行更新，计算状态的导数，产生过零事件。从求解器传递给系统的信息包括时间、输入和当前状态；反过来，系统为求解器提供块的输出、状态的更新和状态的导数。计算连续状态包含两个步骤：首先，求解器为待更新的系统提供当前状态、时间和输出值，系统计算状态导数，传递给求解器；然后求解器对状态的导数进行积分，计算新的状态的值。状态计算完成后，块的输出更新再进行一次。这时，一些块可能会发出过零警告，促使求解器探测出发生过零的准确时间。求解器和系统间的关系如图 9.4 所示。实际上求解器和系统之间的对话是通过不同的标志来控制的。求解器在给系统发送标志的同时也发送数据。系统使用这个标志来确定所要执行的操作，并确定所要返回的变量的值。

仿真运行阶段的工作可以概括为：

(1) 计算输出。

(2) 更新离散状态。

(3) 计算连续状态，连续状态的计算过程：

① 每个块按照预先确定的顺序计算输出。

② 每个块使用当前时间、块的输入和状态计算它的导数。

③ 导数返回给求解器，通过积分得到下一步状态的值。

(4) 计算输出，过零可能被激活。

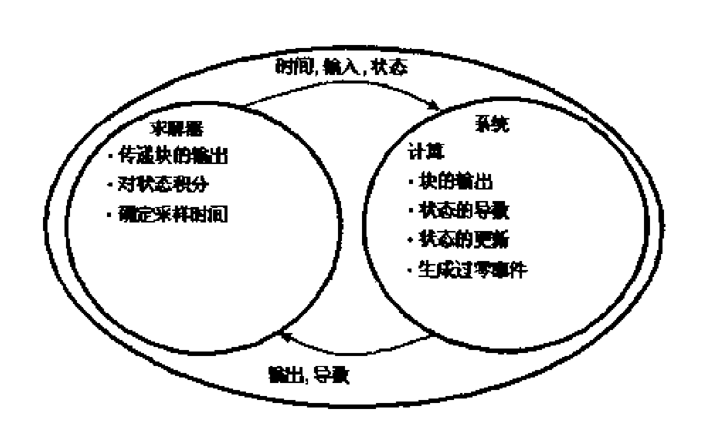


图 9.4　求解器与系统的交互作用关系

9.2.3　S-函数仿真流程

S-函数是 Simulink 的重要组成部分，它的仿真过程包含在 Simulink 仿真过程之中，所以上一节所述同样适用于 S-函数。如图 9.5 所示，S-函数的仿真流程也包括初始化阶段和运行阶段两个阶段。图 9.5 中每个功能模块都对应于一个仿真例程或者回调函数。初始化工作完成以后，在每一个仿真步长（time step）内完成一次求解，如此反复，形成一个仿真循环，直到仿真结束。对于连续状态求解器，一个仿真步长又分为 major time step 和 minor time step，其中 minor time step 是 major time step 的一部分，Simulink 在 minor time step 做数值积分运算。图 9.5 中有两个输出环节和求导数环节，为了提高积分精度，求解器会进行两次输出一致性检查，当两次输出大于求解器页面所设置的误差限（tolerance）时，会以一个小的步长重新计算输出和导数。（进一步的介绍请参考第 6 章高级积分器的概念。）每个仿真循环的最后检查是否有过零事件的发生，一旦检测到过零事件，Simulink 在发生过零事件的变量或者状态的当前值和一个仿真步长之前的值之间进行插值运算，以提高仿真的精度。关于过零的概念请参考第 6 章。

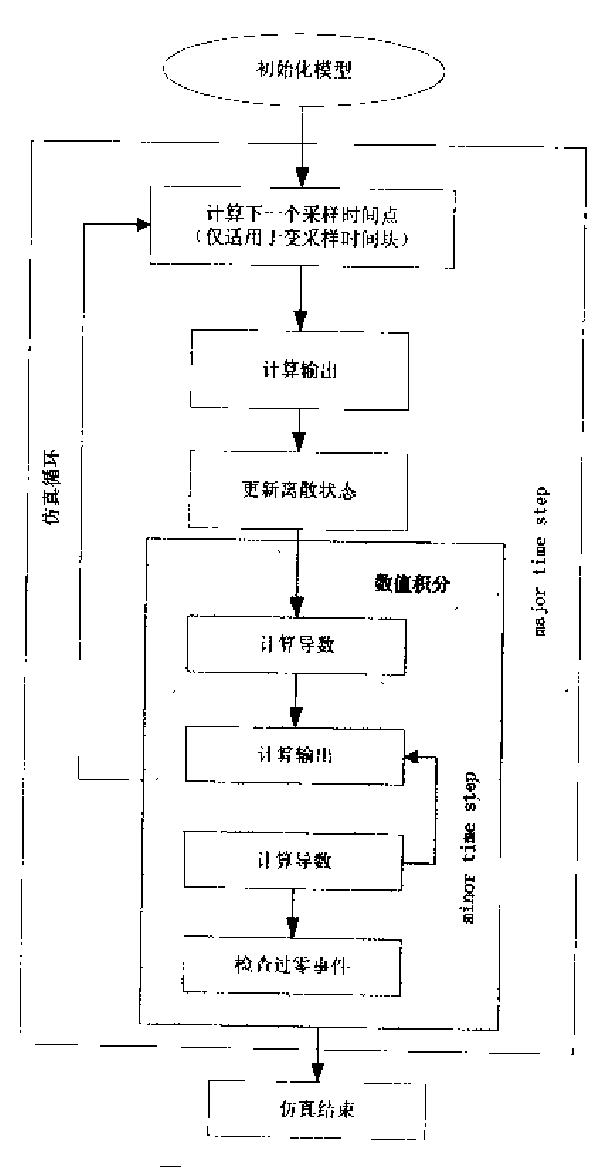


图 9.5　S-函数仿真流程

S-函数依靠回调函数（例程）完成每个仿真阶段的任务。在一次仿真任务中，Simulink 在以下的每个仿真阶段调用相应的 S-函数例程。

(1) 初始化：在仿真开始前，Simulink 在这个阶段初始化 S-函数。

① 初始化结构体 SimStruct，它包含了 S-函数的所有信息。

② 设置输入输出端口数。

③ 设置采样时间。

④ 分配存储空间。

(2) 计算下一个采样时间点：只有在使用变步长求解器进行仿真时，才需要计算下一个采样时间点，即计算下一步的仿真步长。

(3) 计算输出：计算所有输出端口的输出值。

(4) 更新状态：此例程在每个步长处都要执行一次，可以在这个例程中添加每一个仿真步都需要更新的内容，例如离散状态的更新。

(5) 数值积分：用于连续状态的求解和非采样过零点。如果 S-函数存在连续状态，Simulink 就在 minor step time 内调用 mdlDdrivatives 和 mdlOutput 两个 S-函数例程。如果存在非采样过零点，Simulink 将调用 mdlOutput 和 mdlZeroCrossings 例程（过零点检测例程）以定位过零点。

9.3 编写 M 文件 S-函数

限于篇幅，本章仅对最常用的 M 文件 S-函数和 C MEX S-函数做一介绍。由上几节知识可以想象，S-函数无非是由一些仿真功能模块（例程）组成的。这些例程就是 S-函数所特有的语法构成，用户的任务就是在相应的例程内填入合适的代码，供 Simulink 及求解器调用。M 文件 S-函数结构明晰，易于理解，书写方便，且可以调用丰富的 MATLAB 函数，对于一般的应用，使用 MATLAB 语言编写 S-函数就足够了。

9.3.1 M 文件 S-函数的工作流程

M 文件 S-函数和上节所介绍的 S-函数仿真流程是一致的。它调用例程函数的顺序是通过标志 Flag 来控制的。图 9.6 给出了各仿真阶段的标志值、变量值及其对应仿真例程。在

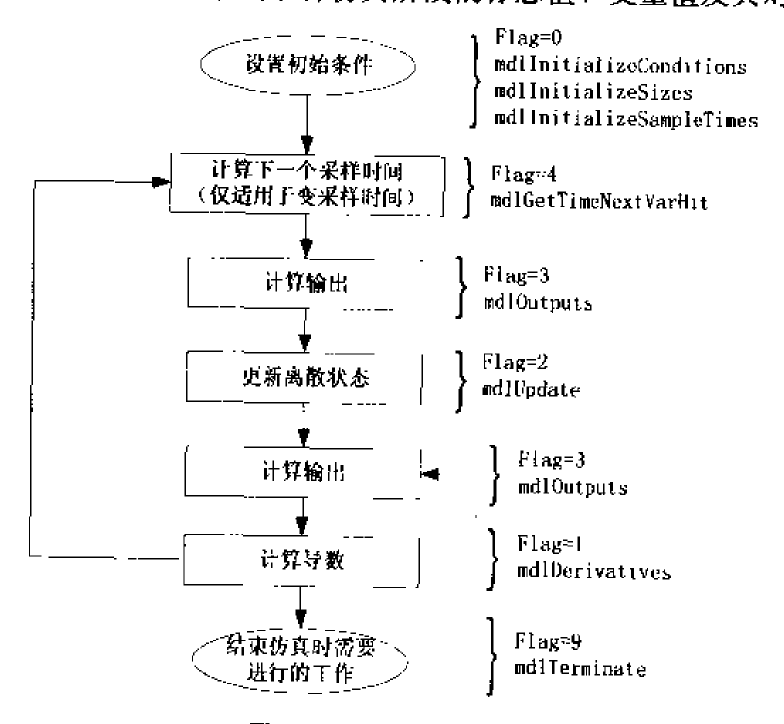


图 9.6 M 文件 S-函数流程

初始化阶段，通过标志 0 调用 S-函数，并请求提供输入输出个数、初始状态和采样时间等信息。然后，仿真开始。下一个标志为 4，请求 S-函数提供下一步的采样时间（这个例程在单采样速率系统下不被调用）。接下来是标志 3，计算块的输出。使用标志 2 更新离散状态。当需要计算状态导数时，给出标志 1。然后求解器使用积分例程计算状态的值。计算状态导数和更新离散状态之后通过标志 3 计算块的输出。这样就完成了一个仿真步长的工作。当到达结束时间时，采用标志 9，做结束前的处理工作。总之，M 文件的 S-函数使用标志变量 Flag 来控制仿真流程。

9.3.2 M 文件 S-函数模板

Simulink 为我们编写 S-函数提供了各种模板文件，其中定义了 S-函数完整的框架结构，用户可以根据自己的需要加以剪裁。编写 M 文件 S-函数时，推荐使用 S-函数模板文件 sfuntmpl.m。这个文件包含了一个完整的 M 文件 S-函数，它包含 1 个主函数和 6 个子函数。在主函数内程序根据标志变量 Flag，由一个开关转移结构（Switch-Case）根据标志将执行流程转移到相应的子函数，即例程函数。Flag 标志量作为主函数的参数由系统（Simulink 引擎）调用时给出。了解这个模板文件的最好方式莫过于直接打开看看其代码。

要打开模板文件，可在 MATLAB 命令行下输入：

```
>>edit sfuntmpl
```

或者双击\S-function demos\M-file S-functions\M-file template 块。

下面是删除了其原有注释的代码，结构反而更为清晰紧凑。

```
% 主函数
% 主函数包含四个输出：sys 数组包含某个子函数返回的值，它的含义随着调用子函数的不同而
% 不同；x0 为所有状态的初始化向量；str 是保留参数，总是一个空矩阵；Ts 返回系统采样时间。
% 函数的四个输入分别为采样时间 t，状态 x，输入 u 和仿真流程控制标志变量 flag，flag 和对应
% 的仿真流程位置如图 9.6 所示；此外输入参数后面还可以接续一系列的附带参数。另外别忘了
% 编写自己的 S-函数时，应该把函数名 sfuntmpl 改为 S-function 块中对应的函数名
function [sys,x0,str,ts] = sfuntmpl(t,x,u,flag)
switch flag
  case 0
    [sys,x0,str,ts]=mdlInitializeSizes;
  case 1
    sys=mdlDerivatives(t,x,u);
  case 2
    sys=mdlUpdate(t,x,u);
  case 3
    sys=mdlOutputs(t,x,u);
  case 4
    sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9
    sys=mdlTerminate(t,x,u);
```

```
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
% 主函数结束
% 下面是各个子函数，即各个仿真例程
% 1. 初始化例程子函数：提供状态、输入、输出、采样时间数目和初始状态的值。必须有该子函
% 数。初始化阶段，标志变量首先被置为 0，S-函数被第一次调用时，mdlInitializeSizes 子函数
% 首先被调用。这个子函数应当为系统提供 S-function 块的下列信息：
% 连续状态的个数
% 离散状态的个数
% 输出的个数
% 输入的个数
% 是否直接馈通：这是一个布尔量，当输出值直接依赖于同一时刻的输入值时为 1；否则为 0
% 采样时间的个数：每个系统至少有一个采样时间
% 这些信息是通过一个数据结构 sizes 来表示的
% 在该函数中用户还应该提供初始状态 x0，采样时间 ts。ts 是一个 m×2 的矩阵，其中第 k 行
% 包含了对应与第 k 个采样时间的采样周期值和偏移量。另外，在该子函数中 str 设置为空：[ ]，
% str 是保留变量，暂时没有任何意义
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;                % 生成 sizes 数据结构
sizes.NumContStates   = 0;       % 连续状态数，缺省为 0
sizes.NumDiscStates   = 0;       % 离散状态数，缺省为 0
sizes.NumOutputs      = 0;       % 输出量个数，缺省为 0
sizes.NumInputs       = 0;       % 输入量个数，缺省为 0
sizes.DirFeedthrough = 1;        % 是否存在直接馈通。1：存在；0：不存在，缺省为 1
sizes.NumSampleTimes = 1;        % 采样时间个数，至少是一个
sys = simsizes(sizes);           % 返回 sizes 数据结构所包含的信息

x0  = [ ];                       % 设置初始状态
str = [ ];                       % 保留变量置空
ts  = [0 0];        % 采样时间：[采样周期 偏移量]，采样周期为 0 表示是连续系统

% 2. 计算导数例程子函数：给定 t,x,u，计算连续状态的导数，用户应该在此给出系统的连续状态
% 方程。该子函数可以不存在
function sys=mdlDerivatives(t,x,u)
sys = [ ];       % sys 表示状态导数，即 dx

% 3. 状态更新例程子函数：给定 t,x,u，计算离散状态的更新。每个仿真步长必然调用该子函数，
% 不论是否有意义。用户除了在此描述系统的离散状态方程外，还可以填入其它每个仿真步长都
```

```
% 有必要执行的代码
function sys=mdlUpdate(t,x,u)
sys = [ ];      % sys 表示下一个离散状态 即 x(k+1)

% 4. 计算输出例程子函数：给定 t,x,u, 计算输出。该子函数必须存在，用户可以在此描述系统的
% 输出方程
function sys=mdlOutputs(t,x,u)
sys = [ ];      % sys 表示下输出，即 y

% 5. 计算下一个采样时间，仅在系统是变采样时间系统时调用
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1;           % 设置下一次的采样时间是 1 s 以后
sys = t + sampleTime;     % sys 表示下一个采样时间点

% 6. 仿真结束时要调用的例程函数，在仿真结束时调用，用户可以在此完成结束仿真所需的必要
% 工作
function sys=mdlTerminate(t,x,u)
sys = [ ];
```

下面我们使用 M 文件 S-函数实现几种不同的系统。

9.3.3 含用户参数的简单系统

M 文件 S-函数除了几个必需的参数，还可以加入用户自定义参数，这些参数需要在 S-函数的输入参数中列出。首先主函数要做适当的修改，以使将用户参数传递到子函数中，子函数的定义也应当进行相应的修改，以便通过输入参数接收用户的参数。在编写 S-函数时，应当区分哪些参数会影响一个子函数的执行，然后针对这些参数做相应的改动。另外，使用这些参数时不要忘了在 S-functions 模块的模块参数对话框内输入它们。

【例 9.2】 用 S-函数实现 gain 模块。

解： 增益值作为 S-函数用户自定义参数由用户输入。

(1) 对 M 文件 S-函数的主函数定义做了修改，增加新的参数，并采用新的函数名：

```
function [sys,x0,str,ts]=sfun_vargain(t,x,u,flag,gain)
```

(2) 由于增益参数只是用来计算输出值，因而对 mdlOutputs 的调用可修改成：

```
case 3
sys=mdlOutputs(t,x,u,gain);
```

(3) 修改初始化例程：

```
sizes.NumContStates  = 1;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
```

```
x0=0;
```

(4) mdlOutputs 子函数的定义也做了相应的修改，将增益作为参数输入：

```
function sys=mdlOutputs(t,x,u,g)
sys = g*u;
```

输出通过增益和输入的乘积得到，并通过 sys 返回。从图 9.7 中可以清晰地看出用户自定义参数 gain 的来龙去脉。图中有两个 S-函数块，一个是原始未封装的 S-函数模块，一种是封装后的 S-函数模块。要封装一个 S-函数应先选中 S-函数块然后点击菜单\Edit\Mask S-functions。封装后的模块只需按照提示填入参数即可，对于没有封装的 S-函数，双击 S-函数后弹出一个对话框，在其中的 S-function parameters 编辑框中填入参数值即可，若有多个参数值时，需要用逗号分开。

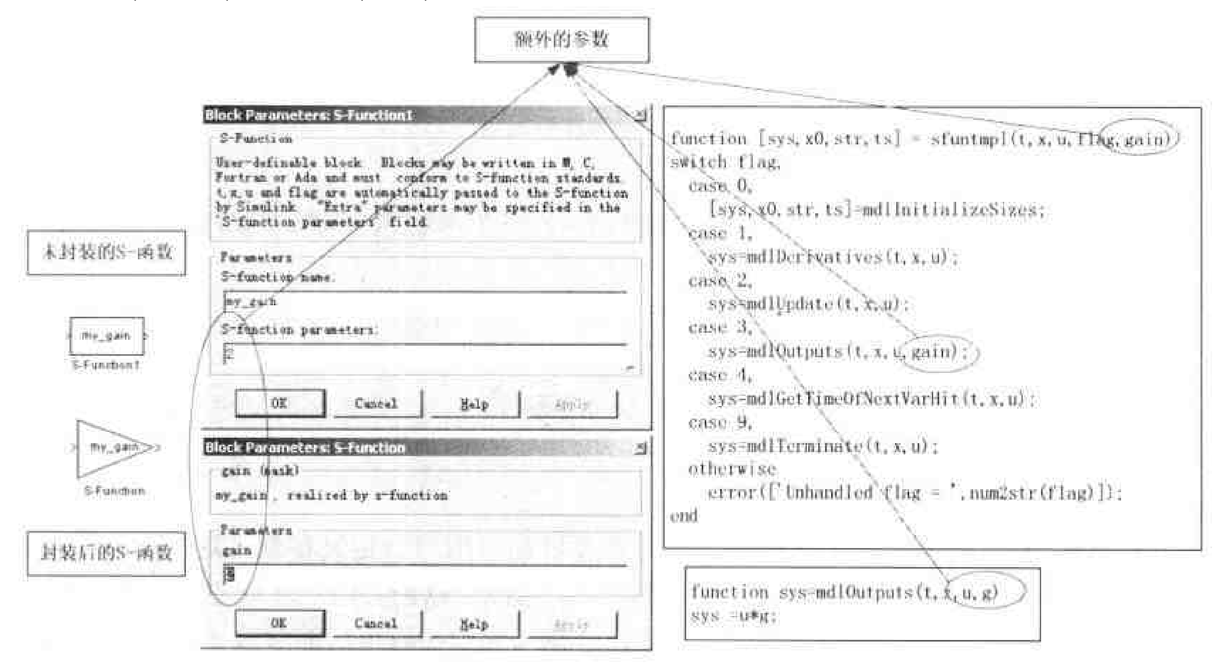


图 9.7 含用户参数的简单系统

9.3.4 离散系统的 S-函数描述

用 S-函数模板实现一个离散系统时，首先对 mdlInitializeSizes 子函数进行修改，声明离散状态的个数，对状态进行初始化，确定采样时间等。然后再对 mdlUpdate 和 mdlOutputs 函数做适当的修改，分别输入要表示的系统的离散状态方程和输出方程即可。

离散系统最简单的例子是单位延迟。单位延迟的差分方程为 $y(k+1)=u(k)$，这相当于在每个仿真步长对采样做一阶保持。等价的状态方程表达为 $x(k+1)=u(k)$，$y(k)=x(k)$。于是，mdlUpdate 和 mdlOutputs 应当修改为：

```
function sys=mdlUpdate(t,x,u)
sys = u;
function sys=mdlOutputs(t,x,u)
sys = x;
```

要观察该 S-函数源文件，可在 MATLAB 命令行输入：

```
>> edit sfundsc2
```

然后双击\S-function demos\M-file S-functions\Unit delay 观察仿真结果。图 9.8 是其 Simulink 框图。示波器显示正弦信号被采样并作一阶保持。用户不妨将 Sine Wave 块 Sample time 参数设置为其它值，再观察一下会有什么结果。

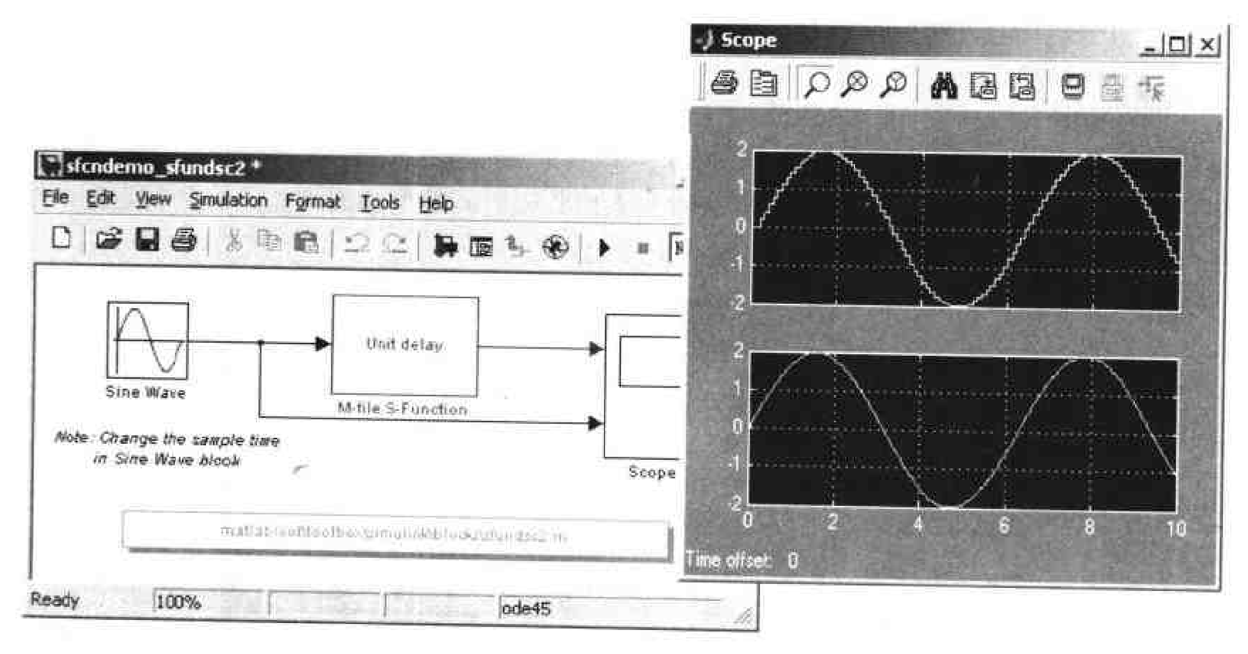


图 9.8　用 S-函数实现单位延迟

【例 9.3】　编写一个 S-函数，说明人口的动态变化。设人口出生率为 r，资源为 K，初始人口数量为 init，则人口变化规律为：p(n)=r*p(n-1)*(1-p(n-1)/K)，p(0)=init。

解：(1) 修改 M 文件 S-函数主函数：

```
function [sys,x0,str,ts]=sfun_population(t,x,u,flag,r,K,init)
case 0,
[sys,x0,str,ts]=mdlInitializeSizes(init);
...
case 2,
sys=mdlUpdate(t,x,u,r,K);
```

(2) 修改初始化部分：

```
function [sys,x0,str,ts]=mdlInitializeSizes(init)
sizes = simsizes;
sizes.NumContStates  =  0;
sizes.NumDiscStates  =  1;     % 一个离散状态，人口数量
sizes.NumOutputs   =  1;       % 一个输出
sizes.NumInputs   =  0;
sizes.DirFeedthrough  =  0;    % 不存在直接馈通
sizes.NumSampleTimes  =  1;
sys = simsizes(sizes);
x0  = init;  % 初始状态：人口基数
...
```

(3) 在这个例子中，p 为状态，输出等于状态，于是

```
function sys=mdlUpdate(t,x,u,r,K)
sys = [r*x*(1-x/K)];
function sys=mdlOutputs(t,x,u)
sys = [x];
```

设置初始人口基数 init=1e5,资源 K=1e6，人口出生率 r=1.05。仿真结果和 Simulink 框图如图 9.9 所示。可以看到在以上条件下人口数量随时间缓慢减少，直到稳定到一个值。当增加资源的量时，例如令 K=3e6，会看到人口数随时间的增加而增加，直到稳定到一个值，说明在此资源数下已经不能够再承载更多的人口。

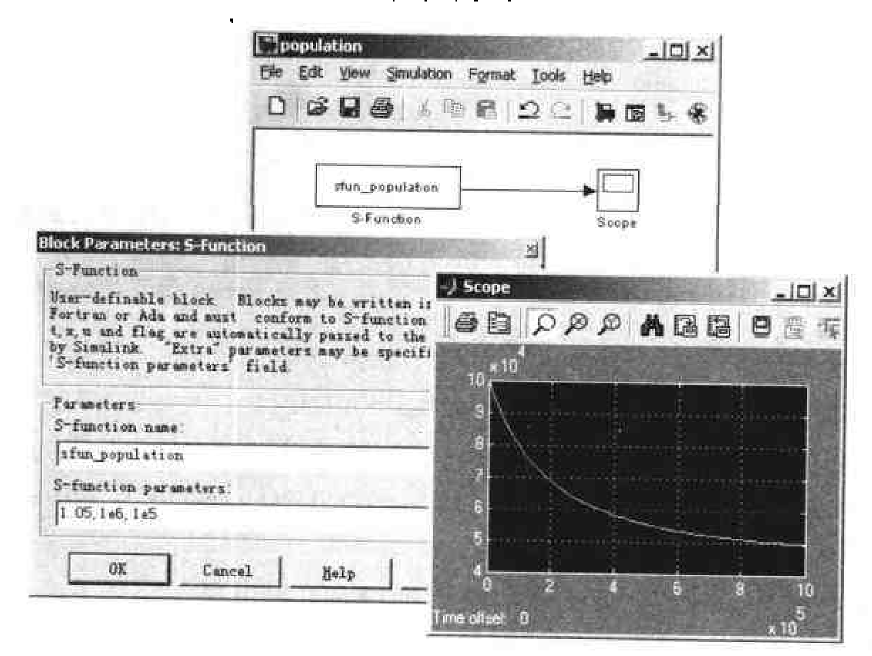


图 9.9 编写 S-函数仿真人口系统

9.3.5 连续系统的 S-函数描述

用 S-函数实现一个连续系统时，首先 mdlInitilizeSizes 子函数应当做适当的修改，包括确定连续状态的个数、状态初始值和采样时间设置。另外，还需要编写 mdlDerivatives 子函数，将状态的导数向量通过 sys 变量返回。如果系统状态不止一个，可以通过索引 x(1)，x(2) 得到各个状态。当然，对于多个状态，就会有多个导数与之对应。在这种情况下，sys 为一个向量，其中包含了所有连续状态的导数。与前文所述一样，修改后的 mdlOutputs 中应包含系统的输出方程。下面使用 S-函数实现一个最简单的连续系统模块——积分器。

【例 9.4】 用 M 文件 S-函数实现一个积分器。

解：对于一个积分器，输入输出之间的关系：$\dot{y}=u$，令状态 $x=u$，则系统状态方程为 $\dot{x}=u$，系统输出方程为 $y=x$。下面修改 S-函数模板文件。

(1) 修改 S-函数模板的第一行：

```
function [sys,x0,str,ts] = sfun_int(t,x,u,flag,initial_state)
```

(2) 初始状态应当传递给 mdlInitializeSizes：

```
case 0,
[sys,x0,str,ts]=mdlInitializeSizes(initial_state);
```

(3) 设置初始化参数：

```
function [sys,x0,str,ts]=mdlInitializeSizes(initial_state)
sizes = simsizes;
sizes.NumContStates  = 1;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;    % 采样时间个数，至少是一个
sys = simsizes(sizes);
x0  = initial_state;         % 初始化状态变量
```

(4) 书写状态方程：

```
function sys=mdlDerivatives(t,x,u)
sys = u;
```

(5) 添加输出方程：

```
function sys=mdlOutputs(t,x,u)
sys = x;
```

令输入 $u = \sin t$，初始状态 init_state = −1，则得到如图 9.10 所示的仿真结果。

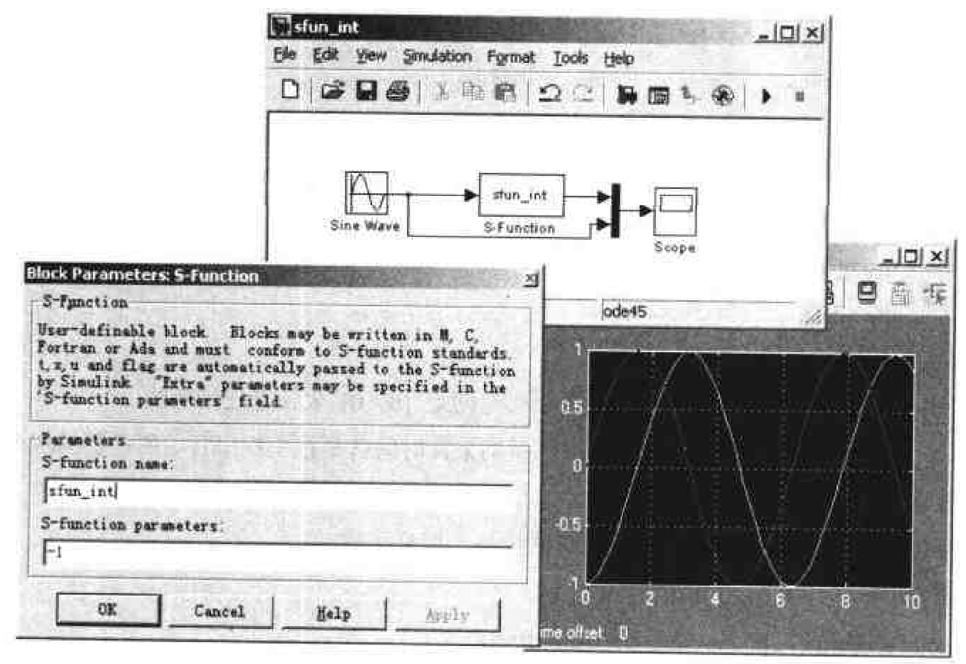


图 9.10　用 S-函数实现的积分器

9.3.6　混合系统的 S-函数描述

所谓混合系统，就是既包含离散状态，又包含连续状态的系统。这里使用一个 Simulink

自带的例子 mixedm.m 来说明这个问题。mixedm.m 描述了一个连续积分系统外加一个离散单位延迟。其 Simulink 框图如图 9.11 所示。在仿真的每个采样时间点上 Simulink 都要调用 mdlUpdate、mdlOutput 和 mdlGetTimeOfNextVarHit 仿真例程（如果是固定步长就不需要 mdlGetTimeOfNextVarHit 例程）；所以在 mdlUpdate、mdlOutput 中需要判断是否需要更新离散状态和输出。因为对于离散状态并不是在所有的采样点上都需要更新，否则就是一个连续系统了。

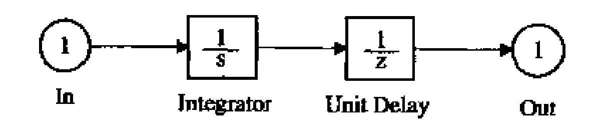


图 9.11 一个简单的混合系统

在命令行输入：

```
>> edit mixedm.m
```

下面列出了部分代码：

```
function [sys,x0,str,ts] = mixedm(t,x,u,flag)
dperiod = 1;
doffset = 0;                    % 设置离散采样周期和偏移量
...                             % switch-case 结构
function [sys,x0,str,ts]=mdlInitializeSizes(dperiod,doffset)
sizes = simsizes;
sizes.NumContStates  = 1;        % 一个连续状态
sizes.NumDiscStates  = 1;        % 一个离散状态
sizes.NumOutputs     = 1;        % 一个输出
sizes.NumInputs      = 1;        % 一个输入
sizes.DirFeedthrough = 0;        % 没有前馈
sizes.NumSampleTimes = 2;        % 两个采样时间
sys = simsizes(sizes);
x0  = ones(2,1);
str = [ ];
ts  = [0 0; dperiod doffset];  % 一个采样时间是 [0  0] 表示是连续系统
                               % 离散系统的采样时间就是在主程序开始所设置的两个变量
function sys=mdlDerivatives(t,x,u)
sys = u;                       % 连续系统是一积分环节

function sys=mdlUpdate(t,x,u,dperiod,doffset)
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys = x(1);                  % 离散系统是一延迟环节
else
  sys = [ ];
```

```
end

function sys=mdlOutputs(t,x,u,doffset,dperiod)
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys = x(2);                 % 输出采样延迟
else
  sys = [ ];
end
```

注意：文中黑体代码表示如果仿真时间在采样点的正负 1e-8 的范围内，更新状态和输出，否则保持不变。

9.4 编写 C MEX S-函数

尽管 M 文件 S-函数非常容易编写和理解，但是有一个主要的缺点：由于在每个仿真步都要激活 MATLAB 解释器，使得仿真的速度变慢。另外，当需要利用 RTW 从 Simulink 框图生成实时代码时，框图中不能包含 M 文件 S-函数。而 C MEX S-函数不仅执行速度快，而且可以用来生成独立的仿真程序。不仅如此，现有的一些用 C 语言编写的程序还可以方便地通过包装程序结合到 C MEX S-函数中。这样的 S-函数由于结合了 C 语言的优势，可以实现对操作系统和硬件的访问，这种特性可以用来实现与串口或网络的通信，以及编写设备的驱动等。总之，用 C 语言编写的 S-函数具有以下优点：

(1) 执行速度快。
(2) 实时代码生成。
(3) 包含已有的 C 代码。
(4) 能够访问操作系统接口。
(5) 可以编写设备驱动。

下面介绍一些用 C 语言编写 S-函数所需的几个基本概念。

9.4.1 MEX 文件

对于 M 文件 S-函数，在 MATLAB 环境下可以通过解释器直接执行，对于 C 文件或其它语言编写的 S-函数，则需要先编译成可以在 MATLAB 内运行的二进制代码：动态链接库或者静态库，然后才能使用，这些经过编译的二进制文件即是所谓的 MEX 文件，在 Windows 系统下 MEX 文件后缀为 dll。要将 C 文件 S-函数编译成动态库，需在 MATLAB 命令行下输入：

```
>> mex my_sfunction.c
```

要使用 mex 命令，首先需要在系统中安装一个 C 编译器。如果还没有设置编译器，则要在命令行下输入：

```
>> mex –setup
```

然后按照提示选取 VC、BC 或者其它的 C 编译器。推荐使用 VC 编译器。生成的文件

my_sfunction.dll 即是我们需要的动态库文件。当 C 文件中使用到其它库文件时，编译时应该在其后加上所需库文件名。如：

```
>> mex my_sfunction.c kernel32.lib
```

有关 Mex 命令更详细的描述请参考 MATLAB 帮助。

其实 M 文件 S-函数也可以编译成 MEX 文件再使用，这时 MATLAB 调用的就是动态链接库而不是 M 文件本身了。M 文件 S-函数编译成 MEX 文件实际上是先用下面的命令将其转换成 C 文件 S-函数，再调用 C 编译器编译链接的。这样编译成的代码一般比 M 文件的 S-函数执行速度快，且其代码由于是二进制的而具有保密性。要将 M 文件编译成 MEX 文件，可在 MATLAB 命令行下输入：

```
>> mcc -S mfilename
```

其中，-S 选项表示编译的是 S-函数。可在 Windows 系统下生成文件 mfilename.c、mfilename.h、mfilename_simulink.c 和 mfilename.dll。在这个过程中，mcc 自动调用了 mex 命令将生成的 C 代码编译成动态链接库。

9.4.2 Simstruct 数据结构

一个称为 SimStruct 的数据结构描述了 S-函数中所包含的系统。此结构在头文件 simstruc.h 中定义。SimStruct 将描述系统的所有信息，即封装系统的所有动态信息。它保存了指向系统的输入、状态、时间等存储区的指针，另外它还包含指向不同 S-函数方法（S-函数例程）的指针。实际上整个 Simulink 框图模型本身也是通过一个 SimStruct 数据结构来描述的，它可以被视为与 Simulink 框图模型等价的表达。SimStruct 有着类似于目录树的数据结构，每个 SimStruct 含有指向它的 Child SimStruct（子 SimStruct 结构体）的指针域。simstruc.h 文件还定义了从 SimStruct 中读取信息和设置变量值的的宏函数，使用这些宏函数我们可以得到时间、状态、输入和参数等信息，或者设置输出、状态导数和状态更新等的值。它们是用户理解 C MEX S-函数的关键，在编写 C MEX S-函数时，会经常遇见这些宏。

在编写 C MEX S-函数时，经常会看见这样的参数：SimStruct *S，这里 S 代表 S-function 块。有关模块的信息都是通过 S 来引用或设置的。

9.4.3 工作向量 （Work Vector）

在仿真过程中不释放的内存区域称之为持续存储区（Persistent Memory Storage），为全局变量或局部静态变量分配的内存就是这样的区域。当一个模型中出现同一个 S-函数的多个实例时，这些全局变量或者局部静态变量就会发生冲突，导致仿真不能正确进行。因为这些实例使用了共同的动态链接库（MEX 文件），正如在 Windows 下多个实例在内存中只有一个映像一样。此时 Simulink 为用户提供了工作向量（work vector）来解决这个问题。工作向量是 Simulink 为每个 S-函数实例分配的持续存储区，它完全可以替代全局变量和局部静态变量。

工作向量支持整数、实数、指针和通用数据等数据类型。表 9.2 列出了分配、设置和读取工作向量的一些常用函数。函数参数 S 是指向与此模块相联系的 SimStruct 结构体的指针。

表 9.2　工作向量函数

函数名称	功 能 描 述	所 在 例 程
ssSetNumRWork	设置为实数型工作向量维数	在 mdlInitializeSizes 例程中分配，在 mdlStart 或者 mdlInitializeConditions 中初始化工作向量
ssSetNumIWork	为整型工作向量维数	
ssSetNumPWork	为指针数据类型工作向量维数	
ssSetRWorkvalue	设置实数型工作向量值	mdlOutputs
ssSetIWorkvalue	设置整型工作向量值	
ssSetPWorkvalue	设置指针型工作向量值	
ssGetRWorkvalue	读取实数型工作向量值	mdlUpdate 或者 mdlderivatives
ssGetIWorkvalue	读取整型工作向量值	
ssGetPWorkvalue	读取指针型工作向量值	

下面说明如何使用工作向量来保存和使用一个指向文件的指针。

(1) 首先在 mdlInitializeSizes 例程中设置指针工作向量的维数：

```
ssSetNumPWork(S, 1)                /*设置指针工作向量维数为 1*/
```

(2) 在 mdlStart 例程中为该指针向量赋值：

```
static void mdlStart(real_T *x0, SimStruct *S)
{
FILE *fPtr;
void **PWork = ssGetPWork(S);
/*获取指向指针工作向量的指针，因为工作向量本身是指针数组，所以这里 Pwork 是指向指针
  的指针*/
fPtr = fopen("file.dat", "r");
PWork[0] = fPtr;  /*将文件指针存入指针工作向量 */
/* ssSetPWorkValue(S,0,fPtr); 显然更为简洁*/
}
```

(3) 在仿真结束时释放文件指针：

```
static void mdlTerminate(SimStruct *S)
{
if (ssGetPWork(S) != NULL) { /*首先判断是否存在指针工作向量*/
    FILE *fPtr;
    fPtr = (FILE *) ssGetPWorkValue(S,0);      /*再判断文件指针是否为空*/
    if (fPtr != NULL)     { fclose(fPtr);      /*关闭文件*/ }
    ssSetPWorkValue(S,0,NULL);                 /*指针工作向量置空*/
    }
}
```

实际上，在编写 S-函数时使用工作变量不仅麻烦而且不符合习惯，如果不需要在同一个 Simulink 模型中多次使用同样的 S-函数，使用全局变量也没什么不好。但是在实践中发

现即使对于不同的 S-函数，当全局变量变量名定义相同，且两个不同的 S-函数在同一个 Simulink 模型中时，也会出现错误，造成仿真无法正常进行。

9.4.4 C MEX S-函数流程

了解 Simulink 如何与 S-函数相互作用完成动态系统的仿真对用户编写 S-函数是非常有帮助的。前面已经对此进行了介绍，不同的是 C MEX S-函数的流程控制更为精细，数据 I/O 也更为丰富，但是这里还有一些前面没有涉及到的内容。

图 9.12 显示了 S-函数的数据交换过程。

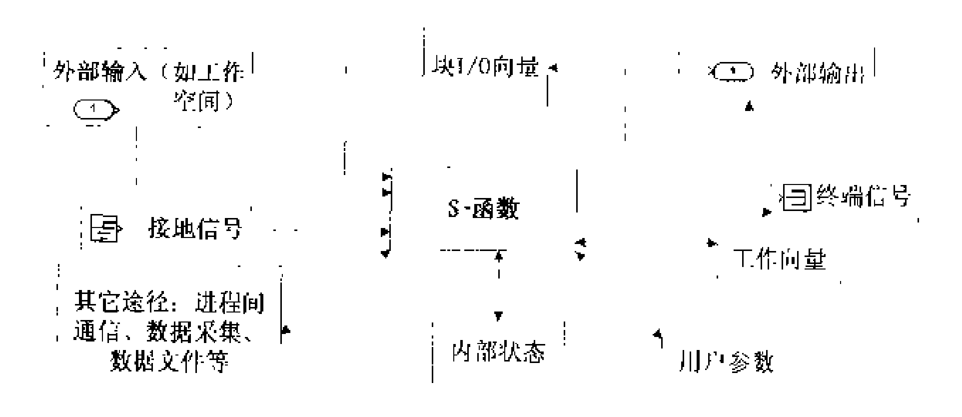


图 9.12 S-函数的数据交换

一个 S-函数有输入、输出、参数和状态、状态导数、工作向量和一些其它的外部输入等数据。通常模块的输入输出都保存在一个块 I/O 向量中，通过访问这些 I/O 向量进行输入与输出。输入输出信号还可能来源于系统的外部输入，用户通过 Simulink 仿真模型的最顶层的 Inport 模块来获取外部的输入(如来源于 MATLAB 工作空间的输入)，或是通过 Outport 块可以向外部输出数据。除此之外，当暂时不需要某些输入输出信号时，但又存在输入输出端口时，可以使用接地信号（ground 块）和终端信号（terminater 块）来防止 Simulink 仿真时出现警告信息。使用 C 语言编写 S-函数的一大好处就是可以丰富系统的 I/O 接口，如可以从文件或者数据采集卡获取数据，或以进程间通信的方式和其它进程交换信息，通过 Socket 用户甚至可以和其它计算机上的 Simulink 模型进行通信。

除了这些输入输出数据外，S-函数还经常用到的内部数据有：

(1) 连续状态。

(2) 离散状态。

(3) 状态导数。

(4) 工作向量。

访问这些数据首先需要一套宏函数获取指向存储它们的存储器的指针，然后通过指针来访问。这时就需要特别注意指针的越界访问问题。还有一项数据就是用户参数，由前文对 M 文件 S-函数的介绍我们知道是通过模块对话框来获得这些参数的。

图 9.13 所示为 Simulink 引擎调用 S-函数回调函数的过程。图中并没有列出全部的 S-函数回调函数，尤其是初始化阶段远比图中所示要复杂。几个必需的例程在图中用实线表示，其它的用虚线表示。

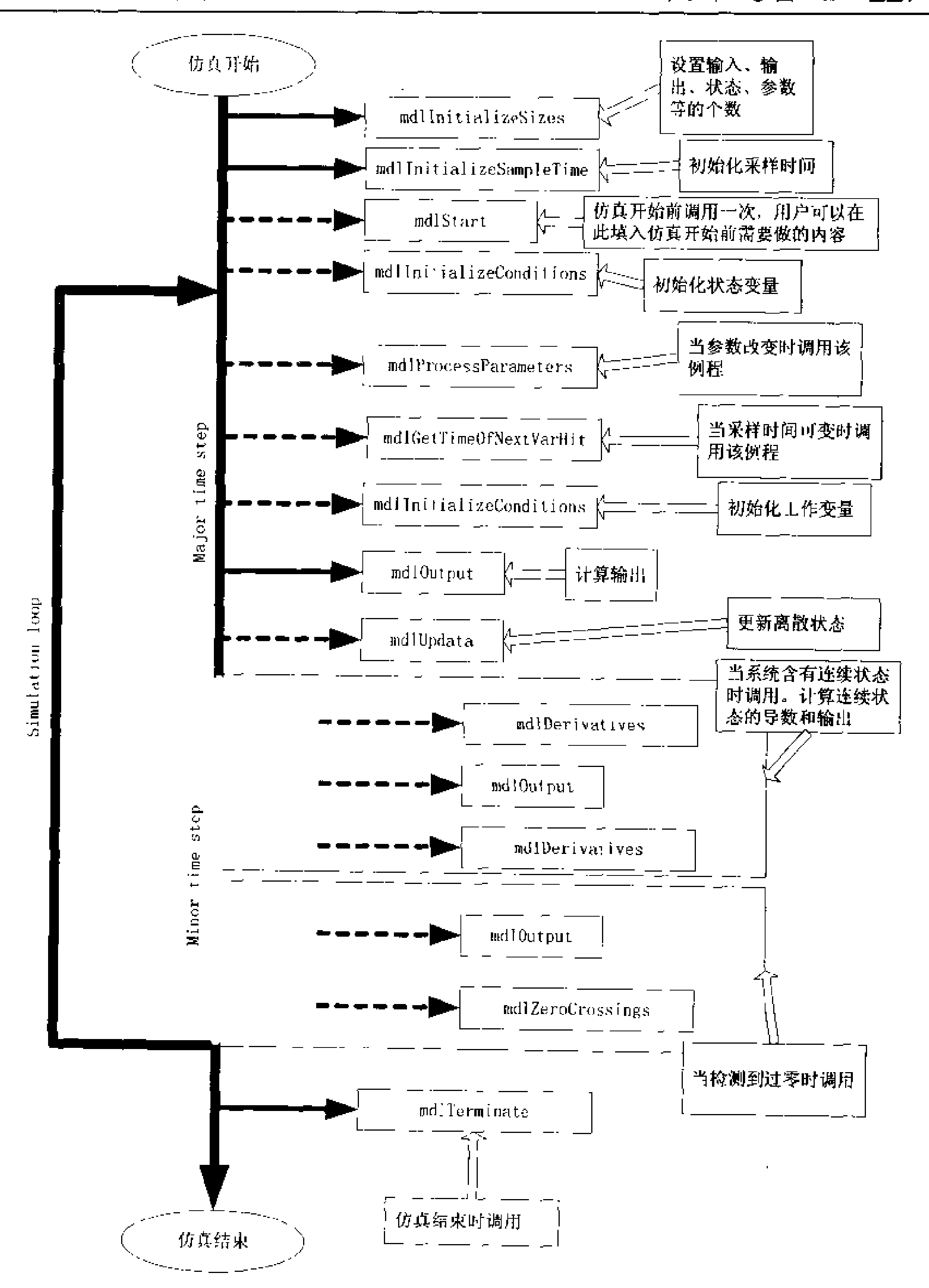


图 9.13　C MEX S-函数工作流程

9.4.5 C MEX S-函数模板

与编写 M 文件 S-函数一样，Simulink 同样为用户提供了编写 C MEX S-函数所需的模板文件。通常使用模板文件 funtmpl_basic.c，它为用户提供了 C MEX S-函数的框架结构。该文件只包含了常用的几个例程，这对于一般的应用已经足够了。文件 sfuntmpl_doc.c 则包含了所有的例程，并附有详细的注释。

每个 C MEX S-函数的开头应包含下列语句：

```
#define S_FUNCTION_NAME your_sfunction_name_here
#define SFUNCTION_LEVEL 2
#include "simstruc.h"
```

其中 *your_sfunction_name_here* 是用户要编写的 S-函数的名字，也就是在 S-Function 块要输入的 S-函数名。S-函数格式随着 Simulink 版本的更替而略有不同，这里 SFUNCTION_LEVEL 就说明了该 S-函数适用的 Simulink 版本。对于 Simulink 4.0 版，SFUNCTION_LEVEL 应该定义为 2。头文件 simstruc.h 则定义了前面所要介绍的一个重要数据结构 SimStruct，simstruc.h 还包含有其它重要的头文件，如 tmwtypes.h，它定义了各种数据类型。

另外在文件的顶部还应包含适当的头文件或定义其它的宏或者变量，就和编写普通的 C 程序一样。在 C MEX S-函数的尾部必然包含下面几行代码：

```
#ifdef MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

宏 MATLAB_MEX_FILE 用于告诉编译器该 S-函数正被编译成 MEX 文件。这几行代码的含义是如果该文件正被编译成 MEX 文件（Windows 下为 dll 文件），包含 Simulink.c；如果正在使用 RTW 将整个 Simulink 框图编译成实时的独立程序（Windows 下为 exe 文件），包含头文件 cg_sfun.h。模板中其它的代码是几个和 M 文件 S-函数中功能类似的 S-函数例程，在此不再赘述。下面介绍用 C MEX S-函数描述动态系统的过程，并在最后给出一个完整的例子。

1. 初始化

C MEX S-函数的初始化部分包含下面三个不同的例程函数：

(1) mdlInitializeSizes：在该函数中给出各种数量信息。

(2) mdlInitializeSampleTimes：在该函数中给出采样时间。

(3) mdlInitializeConditions：在该函数中给出初始状态。

mdlInitializeSizes 通过宏函数对状态、输入、输出等进行设置。工作向量的维数也是在 mdlInitializeSizes 中确定的。与 M 文件 S-函数不同的是，还应该在该例程函数中声明期望的输入参数的个数。表 9.3 列出了初始化所用到的部分宏函数。从表 9.3 中可以看出初始化工作都是通过宏函数访问 SimStruct 数据结构进行的。

表 9.3　S-函数初始化所需宏函数

宏函数定义	功能描述
ssSetNumContStates(S, numContStates)	设置连续状态个数
ssSetNumDiscStates(S, numDiscStates)	设置离散状态个数
ssSetNumOutputs(S, numOutputs)	设置输出个数
ssSetNumInputs(S, numInputs)	设置输入个数
ssSetDirectFeedthrough(S, dirFeedThru)	设置是否存在直接前馈
ssSetNumSampleTimes(S, numSamplesTimes)	设置采样时间的数目
ssSetNumInputArgs(S, numInputArgs)	设置输入参数个数
ssSetNumIWork(S,numIWork)	设置各种工作向量的维数，实际上是为各个工作向量分配内存提供依据。见表 9.7
ssSetNumRWork(S,numIWork)	
ssSetNumPWork(S,numIWork)	

2. 用输入和输出

在 C MEX S-函数中，同样可以通过描述该 S-函数的 SimStruct 数据结构对输入输出进行处理。在 C MEX S-函数中，当需要对一个输入进行处理时，使用宏：

input=ssGetInputPortRealSignalPtrs(S,index)

返回值中含有指向输入向量的指针，其中的每个元素通过*input[i]来访问。指向输出向量的指针通过宏函数 output=ssGetOutputPortRealSignal(S,index)得到。如果想知道输出信号的宽度，使用宏：width=ssGetOutputPortWidth(S,index)；如果需要获得指向输入值的指针，使用宏：ssGetInputPortRealSignalPtrs(S,input_index)；如果需要获得指向输出值的指针，使用宏：ssGetOutputReaSignal(S, output_index)。表 9.4 列出了输入输出相关宏函数。

注意：与 MATLAB 语言不同，在 C 语言中对整个向量进行操作时，必须采用 for 循环对每个元素进行处理。

表 9.4　输入输出相关宏函数

宏函数	功能描述
ssGetInputPortRealSignalPtrs	获得指向输入的指针（double 类型）
ssGetInputPortSignalPtrs	获得指向输入的指针（其它数据类型）
ssGetInputPortWidth	获得指向输入信号宽度
ssGetInputPortOffsetTime	获得输入端口的采样时间偏移量
ssGetInputPortSampleTime	获得输入端口的采样时间
ssGetOutputPortRealSignal	获得指向输出的指针
ssGetOutputPortWidth	获得指向输出信号宽度
ssGetOutputPortOffsetTime	获得输出端口的采样时间偏移量
ssGetOutputPortSampleTime	获得输出端口的采样时间

例如：下面的一段代码获得指向输入和输出的指针，然后把输入乘以 5 后送往输出。

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T              i;
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T             *y = ssGetOutputPortRealSignal(S,0);
    int_T              width = ssGetOutputPortWidth(S,0);
    for (i=0; i<width; i++) {
        *y++ = 5 *(*uPtrs[i]);
    }
}
```

3. 使用参数

使用用户自定义参数时，在初始化中必须说明参数的个数。为了得到指向存储参数的数据结构的指针，使用宏：ptr=ssGetSFcnParam(S, index)；为了得到存储在这个数据结构中指向参数值本身的指针，使用宏：mxGetPr(ptr)；使用参数值时使用宏：param_value=*mxGetPr(ptr)。下面的代码首先获取参数 gain 的值，然后输入乘以 gain 作为输出：

```
#define GAIN=ssGetSFcnParam(S,0)
static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T              i;
    InputRealPtrsType  uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T             *y  = ssGetOutputPortRealSignal(S,0);
    int_T              width = ssGetOutputPortWidth(S,0);
    real_T             gain = *mxGetPr(GAIN);for (i=0; i<width; i++) {
        *y++ = gain *(*uPtrs[i]);
    }
}
```

4. 使用状态

如果 S-函数包含连续的或离散的状态，则需要编写 mdlDerivatives 或 mdlUpdate 子函数。若要得到指向离散状态向量的指针，使用宏：ssGetRealDiscStates(S)；若要得到指向连续状态向量的指针，使用宏：ssGetContStates(S)；在 mdlDerivatives 中，连续状态的导数应当通过状态和输入计算得到，并将 SimStruct 结构体中的状态导数指针指向得到的结果，这通过下面的宏完成：*dx=ssGetdX(S)，然后修改 dx 所指向的值。在多状态的情形下，通过索引得到 dx 中的单个元素。它们被返回给求解器通过积分求得状态。需要注意的是，在离散系统中，没有对应于 dx 的变量，由于状态是由 S-函数来更新的，不需要求解器做任何额外的工作。

下面的一段代码描述了连续状态方程：

$$\dot{x}_0 = (1-\alpha x_1)x_0$$
$$\dot{x}_1 = (-1+\beta x_0)x_1$$

```
static void mdlDerivatives(SimStruct *S)
  {
      real_T              alpha = .01;
      real_T              beta = .02;
      real_T              *dx = ssGetdX(S);
      real_T              *x = ssGetContStates(S);
      dx[0] = (1 - alpha*x[1])*x[0];
      dx[1] = (-1 + beta*x[0])*x[1];
  }
```

下面的一段代码描述了离散状态方程：

$$x_0(k+1) = (1-\alpha x_1(k))x_0(k)$$
$$x_1(k+1) = (-1+\beta x_0(k))x_1(k)$$

```
static void mdlUpdate(SimStruct *S, int_T tid)
{
real_T tempX[2] = {0.0, 0.0};
real_T *x = ssGetRealDiscStates(S);
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
UNUSED_ARG(tid); /* not used in single tasking mode */
tempX[0]= (1 - alpha*x[1])*x[0];;
tempX[1]= (-1 + beta*x[0])*x[1];
x[0]=tempX[0];
x[1]=tempX[1];
}
```

【例 9.5】 S-函数 csfunc 描述了一个用状态方程表示的线性连续系统：

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

并通过共享内存与其它程序交换数据。下面是该系统 C MEX S-函数的完整源代码和注释。

```
#define S_FUNCTION_NAME csfunc   /* S-函数名 */
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#include "windows.h"                   /*创建共享内存所需头文件*/
#define U(element) (*uPtrs[element])   /* 宏定义，方便对输入的索引*/
```

```
/* 定义状态方程 A B C D 阵*/
static real_T A[2][2]={ { -0.09, -0.01 } ,
                        {  1   ,  0    }      /*在 C S-函数中有一套自己的数据*/
};                      /*类型表示方法, real_T 表示双精    */
static real_T B[2][2]={ {  1   , -7    } ,    /*度,int_T 表示整型。通用的 C*/
                        {  0   , -2    }      /* 语言数据类型标识同样适用*/
                      };
static real_T C[2][2]={ {  0   , 2     } ,
                        {  1   , -5    }
                      };
static real_T D[2][2]={ { -3   , 0     } ,    /*注意，存在馈通*/
                        {  1   , 0     }
                      };
double *psharedmem;    /* 指向共享内存存储区的指针 */
HANDLE hfilemap;      /* 共享内存句柄 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);  /* 不含用户参数，所以参数个数设为 0 */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }

    ssSetNumContStates(S, 2);        /*系统有两个连续状态*/
    ssSetNumDiscStates(S, 0);        /*系统没有离散状态*/
if (!ssSetNumInputPorts(S, 1)) return; /* 如果设置输入端口数为 1 失败，返回 */
/*S-functions 块只有一个输入端口，当需要多个输入时，使用 mux 模块把需要输入的信号集中
/*成一个向量*/
    ssSetInputPortWidth(S, 0, 2);               /*输入信号宽度为 2*/
    ssSetInputPortDirectFeedThrough(S, 0, 1);   /*设置馈通标志为 1*/
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 2);              /*输出信号宽度为 2*/
    ssSetNumSampleTimes(S, 1);                  /*1 个采样时间*/
    ssSetNumRWork(S, 0);                        /*未使用工作向量 */
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
```

```
static void mdlInitializeSampleTimes(SimStruct *S)
{
ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
/* 连续系统设采样时间为 0，等同于 ssSetSampleTime(S, 0, 0);*/
    ssSetOffsetTime(S, 0, 0.0); /*偏移量为 0*/
}

#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
{
    real_T *x0 = ssGetContStates(S);      /*获得指向连续状态的指针*/
    int_T   lp;
    for (lp=0;lp<2;lp++) {
        *x0++=0.0;                        /*各状态初始化为 0*/
    }
    /*创建共享内存 */
    hfilemap=OpenFileMapping(
                                FILE_MAP_WRITE ,
                                false,
                               "sharedmem"
                                );
/* 获得指向共享内存的指针*/
psharedmem=(double*)MapViewOfFile(hfilemap,FILE_MAP_WRITE,0,0,2*sizeof(double));
}

static void mdlOutputs(SimStruct *S, int_T tid)
{  /* 获得指向输出向量、连续状态向量和输入端口的的指针*/
    real_T          *y     = ssGetOutputPortRealSignal(S,0);
    real_T          *x     = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
     UNUSED_ARG(tid);
    /* y=Cx+Du */    /* 输出方程*/
    y[0]=C[0][0]*x[0]+C[0][1]*x[1]+D[0][0]*U(0)+D[0][1]*U(1);
    y[1]=C[1][0]*x[0]+C[1][1]*x[1]+D[1][0]*U(0)+D[1][1]*U(1);
    psharedmem[0]=y[0]; /* 将输出放到共享内存中供其它进程使用*/
    psharedmem[1]=y[1];
}
```

```
#define MDL_DERIVATIVES
static void mdlDerivatives(SimStruct *S)
{
    real_T              *dx   = ssGetdX(S); /*获得指向状态导数向量的指针*/
    real_T              *x    = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    /* xdot=Ax+Bu */ /*连续状态方程 */
    dx[0]=A[0][0]*x[0]+A[0][1]*x[1]+B[0][0]*U(0)+B[0][1]*U(1);
    dx[1]=A[1][0]*x[0]+A[1][1]*x[1]+B[1][0]*U(0)+B[1][1]*U(1);
}

static void mdlTerminate(SimStruct *S)
{
    UNUSED_ARG(S);
    UnmapViewOfFile(psharedmem); /*在仿真结束时，释放共享内存 */
}

#ifdef  MATLAB_MEX_FILE     /* 是否编译成 MEX 文件 */
#include "simulink.c"       /* 其中包含 MEX 文件的接口方法 */
#else
#include "cg_sfun.h"        /* 代码生成注册函数 */
#endif
```

由以上程序可以看出，C MEX S-函数是通过一套宏函数获得指向存储在 SimStruct 中的输入、输出、状态、状态导数向量的指针来引用输入输出状态等变量的，从而完成对系统的描述。

9.4.6 S-函数包装程序

当用户需要将已有的程序、算法集成到 Simulink 框图模型中时，通常使用 S-函数包装程序（MEX S-function Wrappers）来完成这个任务。所谓的 S-函数包装程序就是一个可以调用其它模块代码的 S-函数，实际上就是通过 S-函数的形式来调用其它语言（MATLAB 语言除外）编写的程序。使用外部模块时，需要在 S-函数中将已有的代码声明为 extern（即外部函数），并且还必须在 mdlOutputs 例程中调用这些已有的代码。为什么必须在 mdlOutput 例程中调用呢？因为只有在 mdlOutput 中才能实现模块输出的更新，从此可以看出包装实际上就是用 S-函数来封装或者包装已有的算法，并将其作为一个模块在 Simulink 中使用。但是这样做存在一个问题：Simulink 在与 S-函数交互作用时，不仅需要调用 mdlOutputs 例程，还需要调用其它几个必需的例程，如 UpDate 例程，即使这个例程中什么也没有；而用户使用 S-函数的目的仅仅是想利用已有的程序，这种多余的调用无疑影响了仿真的效率。解决这个问题的办法是使用 TLC S-Function Wrapper，有关内容请读者参考相关 MATLAB 文献。

下面是一个简单的例子如下。C 源文件：

```
/*wrapfcn.c*/
double wrapfcn(double input)
{
    return(input * 2.0);
}
```

C MEX S-函数源文件如下：

```
#define S_FUNCTION_NAME wrapsfcn
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
extern real_T wrapfcn( (real_T u); /*声明函数为外部函数*/
...
static void mdlOutputs(SimStruct *S, int_T tid)
{
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
real_T *y = ssGetOutputPortRealSignal(S,0);
*y = wrapfcn (*uPtrs[0]);   /*在 mdlOutputs 例程中调用外部函数 wrapfcn*/
}
```

编译带有外部函数的 S-函数时，只需将已有程序的源文件加在 S-函数源文件的后面即可，如下所示：

```
>> mex wrapsfcn.c wrapfcn.c
```

9.4.7 S-function Builder

Simulink 为用户编写常用的 C MEX S-函数提供了方便的开发工具 S-function Builder。使得用户无需了解众多的宏函数就可以编写出自己的 S-函数，只要在对应的位置填入所需的信息和代码，S-function Builder 就会自动生成 C MEX S-函数源文件，并且编译起来也非常方便，只需单击 Build 按钮，就会生成用户所需要的 MEX 文件。

在 Simulink 库浏览器中双击 S-function Builder 块图标，即可打开如图 9.14 所示的 S-function Builder 界面。很容易看出 1、3、4、5 选项卡对应着 S-函数的四个最常用的例程。打开 S-function Builder 为用户生成的 C 源文件，就会发现在各个页面填入的信息和代码被放入了对应的例程中。下面给出用户使用 S-function Builder 编写 S-函数的步骤。

(1) 首先在 S-function name 编辑栏里填入 S-函数名。

(2) 如果存在用户参数，在 S-function parameters 栏填入用户参数缺省值。

(3) 在图 9.14 所示的 S-function Builder 的 Initialization 页中按照提示填入仿真相关信息。这里需要指出的是，Continuous states IC 和 Discrete states IC 分别指的是连续状态初值和离散状态初值。

(4) 在 Libraries 选项卡中填入所需要的库文件（包括目录）、要包含的头文件，以及外部函数声明。如图 9.15 所示。

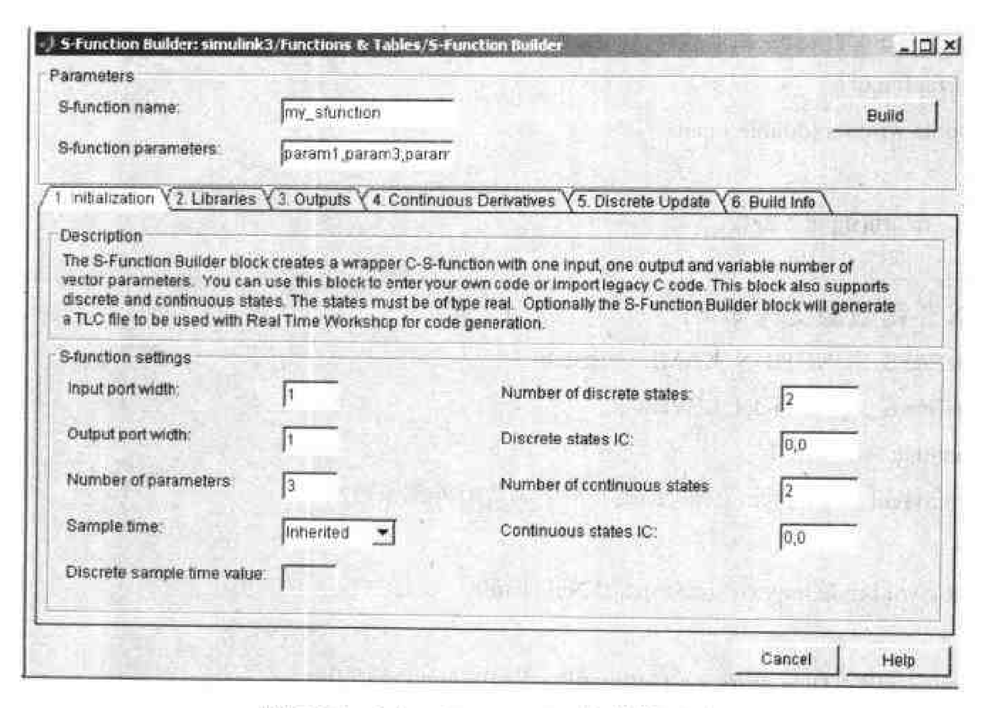


图 9.14　S-function Builder 初始化界面

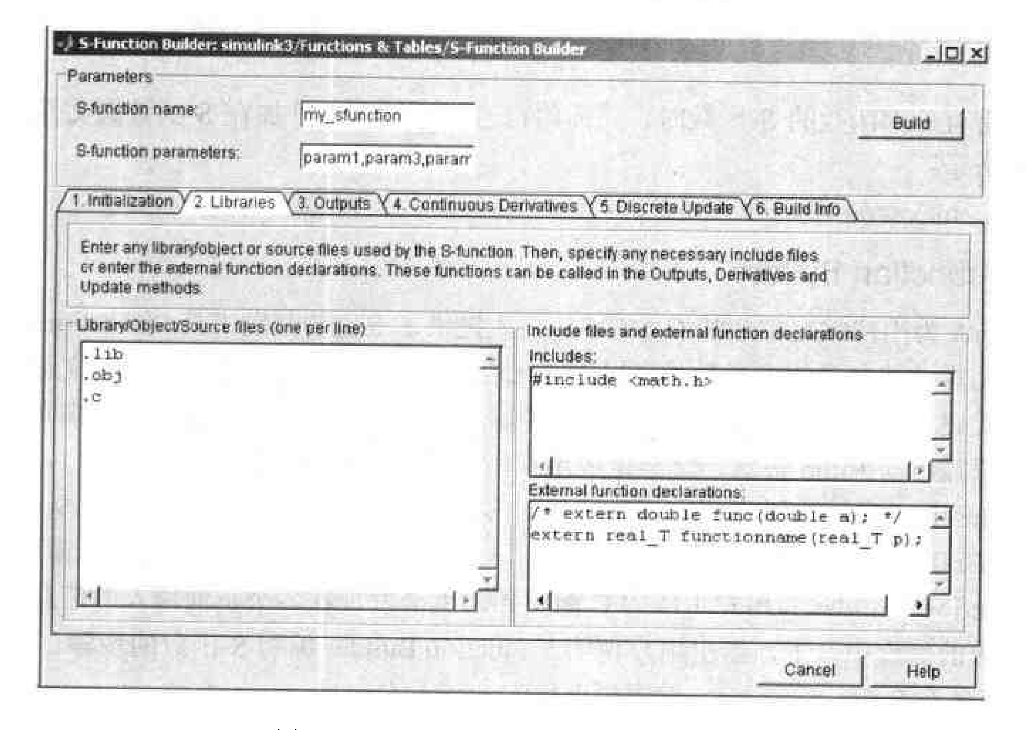
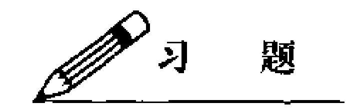


图 9.15　S-function Builder 库文件选项卡

(5) 在 Outputs、Continues Derivatives 和 Discrete Update 页面填入输出方程、连续状态方程和离散状态方程，以及其它用户定制代码。

(6) 单击 Build 按钮，完成生成 C 代码、编译链接等工作。



习　题

1. 如图 9.16 所示的倒立摆系统。长度 l=1 m、质量 m=0.1 kg 的倒立摆用铰链安装在质量 M =1 kg 的小车上，在水平方向上施加控制力 u，相对参考系产生位移 z，忽略各种摩擦、设重力加速度 g =9.81 m/s^2。试用 C MEX S-函数建立上述系统的 Simulink 模型并仿真。

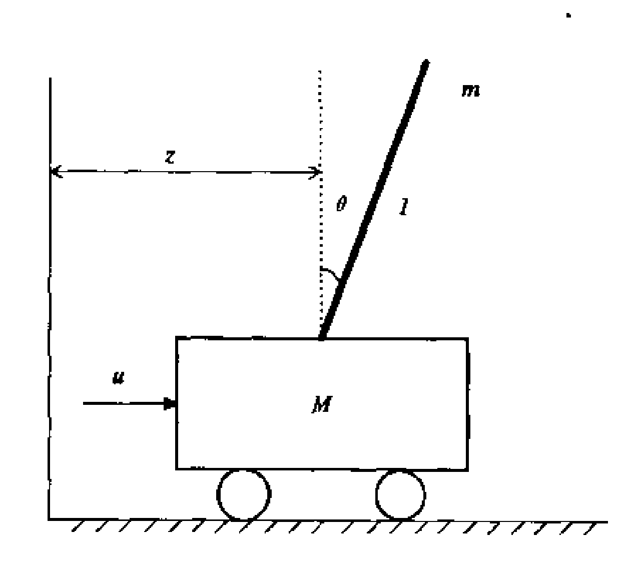


图 9.16　倒立摆系统

提示：首先建立系统的数学模型。设小车位置为 z，则倒立摆质心唯一且为 $z+l\sin\theta$，在力 u 的作用下，小车和倒立摆产生加速运动。根据牛顿第二定律，水平方向的惯性力和 u 平衡：

$$M\frac{\mathrm{d}^2 z}{\mathrm{d}t^2}+m\frac{\mathrm{d}^2}{\mathrm{d}t^2}(z+l\sin\theta)=u$$

即

$$(M+m)\ddot{z}+ml\ddot{\theta}\cos\theta-ml\dot{\theta}^2\sin\theta=u$$

倒立摆绕铰链旋转的惯性力力矩与重力矩平衡：

$$[m\frac{\mathrm{d}^2}{\mathrm{d}t^2}(z+l\sin\theta)]\cdot l\cos\theta=mgl\sin\theta$$

即

$$\ddot{z}\cos\theta+l\ddot{\theta}\cos^2\theta-l\dot{\theta}^2\sin\theta\cos\theta=g\sin\theta$$

联立求解可得

$$\begin{cases}\ddot{z}=\dfrac{(u+ml\dot{\theta}^2\sin\theta)\cos\theta-(g\sin\theta+l\dot{\theta}^2\sin\theta\cos\theta)m}{M\cos\theta}\\[2ex]\ddot{\theta}=\dfrac{(M+m)(g\sin\theta+l\dot{\theta}^2\sin\theta\cos\theta)-\cos\theta(u+ml\sin\theta)}{Ml\cos^2\theta}\end{cases}$$

选取状态 $x=[z\ \dot{z}\ \theta\ \dot{\theta}]^T$，输出 $y=[z\ \theta]$。可得系统状态方程和输出方程：

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \frac{(u+mlx_4^{\ 2}\sin x_3)\cos x_3-(g\sin x_3+lx_4^{\ 2}\sin x_3\cos x_3)m}{M\cos x_3}$$

$$\dot{x}_3 = x_4$$

$$\dot{x}_4 = \frac{(M+m)(g\sin x_3+lx_4^{\ 2}\sin x_3\cos x_3)-\cos x_3(u+ml\sin x_3)}{Ml\cos^2 x_3}$$

$$y=[x_1\ x_3]$$

利用上述状态方程，读者就可以建立倒立摆系统的 S-函数了。上述方程均为非线性方程，为简化起见，读者可以设 $\theta\to 0$，然后简化系统得到系统的线性方程。

2. 使用 S-函数实现一个蹦极跳系统。状态方程为

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = g+\frac{1}{m}bx_1-\frac{a_1}{m}x_2-\frac{a_2}{m}|x_2|x_2$$

输出方程为

$$y=d-x_1$$

其中

$$b(x_1)=\begin{cases}-kx_1, & x_1>0\\ 0, & x_1\leqslant 0\end{cases}$$

d 为人离地面距离。给定的常数为：k=5，a_1=1，a_2=1，g=10。试编写 S-函数描述蹦极跳系统，用户可以输入长度、质量和人离地面的距离。

提示：参考【例 9.3】。

第四部分

实例分析

利用 Simulnik 进行系统设计与分析

前面对使用 Simulink 进行系统模型建立、动态仿真与分析进行了详细的介绍。第四部分将以实际的动态系统设计分析为例，向读者展现 Simulink 的强大功能。本部分包括如下内容:

➢ 控制系统设计与分析:主要介绍在 Simulink 中使用控制系统工具箱进行控制系统的设计与分析。

➢ DSP Blockset:主要介绍基于帧的数字信号处理、滤波器设计与功率谱分析。

第 10 章 控制系统设计分析

内容概要

· LTI Viewer 的使用
· 线性控制系统的设计
· 非线性控制系统的设计

第 1 章至第 8 章详细介绍了使用 Simulink 建立动态系统模型、设置系统模型参数、运行系统仿真以及进行动态系统分析等各项技术。在掌握前面的内容之后，用户便可以得心应手地使用 Simulink 的强大功能来进行各种动态系统的设计、仿真与分析。

本章将以控制系统设计分析为例进行说明。选择控制领域中的系统设计分析为例，其主要原因是 Simulink 对控制领域的系统设计提供了非常强大的设计开发工具，同时控制系统的应用非常广泛，能够被不同领域的系统设计者所接受和理解。

10.1 控制系统的线性分析

在控制系统中，线性系统的设计与分析技术已经比较完善，其应用也非常广泛。虽然在大多数的情况下，实际的系统都是非线性系统，但是在系统的设计与实现时，用户一般总是通过某种方法使用线性系统近似地取代非线性系统。本节将以滑艇速度控制为例说明如何使用 Simulink 对控制系统进行线性分析。

10.1.1 滑艇动态方程及其线性化

1. 滑艇动力学方程

在滑艇的运行过程中，滑艇主要受到如下作用力的控制：滑艇自身的牵引力 F，滑艇受到的水的阻力 f。其中水的阻力 $f=v^2-v$，v 为滑艇的运动速度。由运动学的相关定理可知，整个滑艇系统的动力学方程为

$$\dot{v}=\frac{1}{m}(F-(v^2-v))$$

其中 m 为滑艇的质量。由滑艇系统的动力学方程易知，此系统为一非线性系统。下面来建立此系统的 Simulink 模型并进行线性分析。

2. 滑艇速度控制系统的模型建立与仿真

使用下面的 Simulink 模块建立滑艇速度控制系统的模型：

(1) Sources 模块库中的 Step 模块：用来产生滑艇的牵引力。

(2) Subsystems 模块库中的 Subsystem 模块：构成滑艇速度控制器子系统。

(3) Sinks 模块库中的 Scope 模块：输出滑艇的速度。

(4) Functions & Tables 模块库中的 Fcn 模块：求取水的阻力。

(5) 其它模块：Math 模块库中的 Gain 模块、Continuous 模块库中的 Integrator 模块。

使用 Simulink 建立的系统模型框图如图 10.1 所示。

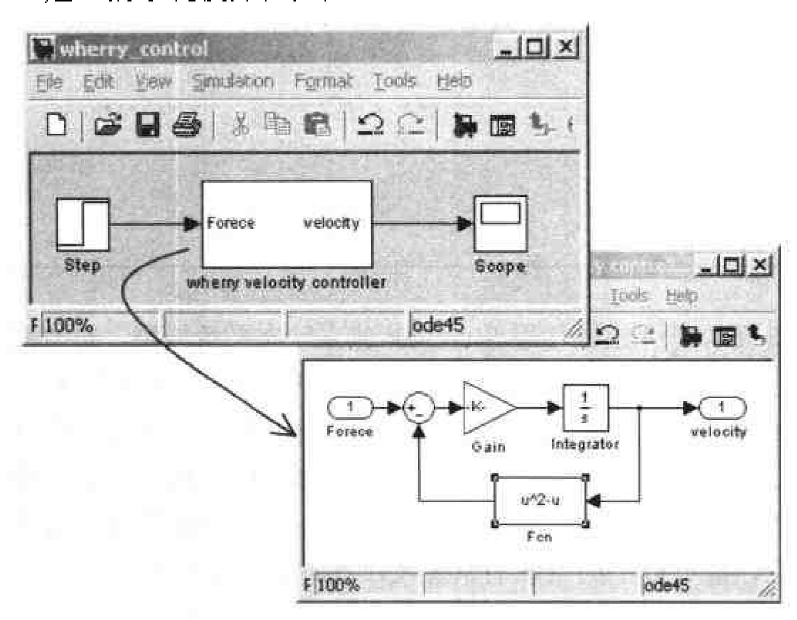
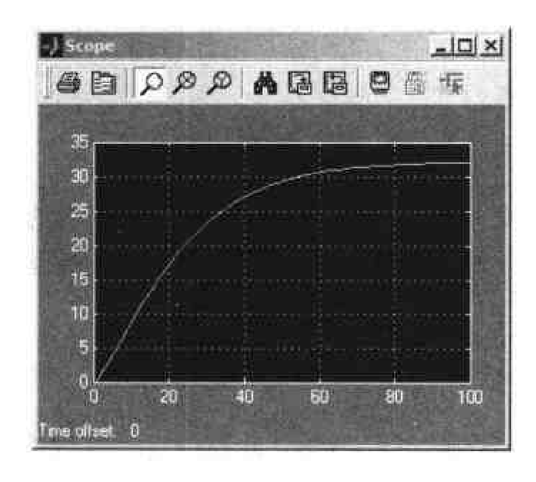


图 10.1 滑艇速度控制系统模型框图

然后设置正确的系统模型参数与仿真参数对此系统进行仿真，其中 Step 的 Final Value 值设置为 1000(即滑艇牵引力 $F=1000$)、子系统中增益模块 Gain 的取值为 1/1000(即 $1/m$)、Fcn 模块的 expression 设置为 u^2-u（求取水的阻力）、系统仿真时间为 0 至 100 s。图 10.2 为系统仿真的结果。



滑艇在牵引力（值1000）的作用下，速度在经过80 s左右的时间后，由0上升并稳定在33 km/h

图 10.2 滑艇系统仿真结果

3. 滑艇速度控制器系统的线性化

对于滑艇速度控制器系统而言，如果要在比赛中获得胜利，则滑艇必须在尽可能短的时间内达到最大速度。设此速度控制器所能达到的最大速度为 100 mph（miles per hour，英

里每小时)。而在前面所提供的滑艇牵引力仅为 1000，故需要设置合适的牵引力对速度控制器进行操纵。为使滑艇速度控制系统的实现变得比较容易，需要设计相应的线性控制器对滑艇的速度进行控制，使其在某个工作点附近与原来滑艇速度控制器的作用基本一致。

在对非线性系统进行线性化处理之前，最好对非线性系统中的系统状态有一个清楚的认识。这是因为在非线性系统的线性化中，需要使用系统的状态。对于滑艇速度控制系统，使用如下的 MATLAB 命令（其中 wherry_control 为系统模型框图的文件名，无后缀 mdl）以获得系统中状态变量的有关信息：

```
>>[sizes,x0,xord]=wherry_control    %获得系统状态变量的信息
sizes =
            1          %表示滑艇速度控制系统中存在一个连续状态
            0
            0
            0
            0
            0
            1
x0 =                   %状态变量的初始值
            0
xord =                 %产生状态变量的系统模块
        'wherry_control/wherry velocity controller/Integrator'
```

由如上的结果可知，在滑艇速度控制系统中仅存在一个连续状态变量，它是由系统模型中的子系统 wherry velocity controller 下的 Integrator 积分模块所产生的；至于返回变量 sizes 中其它数据的含义请参考第 7 章中的介绍，这里不再赘述。

既然滑艇速度最大值为 100 mph，因此在对滑艇速度控制系统进行线性化时，希望此系统能够使滑艇的速度基本稳定在最大速度处。换句话说，系统的工作点应该选择为使速度达到 100 mph 时的系统输入与系统状态。由于对非线性系统进行线性化表示需要给出系统所在的操作点（即平衡点），因此在对滑艇速度控制系统进行线性化之前，需要获得滑艇速度稳定在 100 mph 处的系统平衡点。按照如下步骤可以获得滑艇速度控制系统的平衡点：

(1) 修改系统模型，如图 10.3 所示。

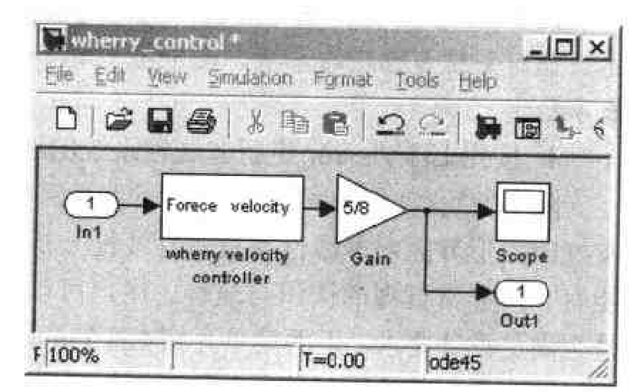


图 10.3　修改之后的模型框图

其中 Inport、Outport 分别表示系统的输入与输出，增益模块的作用是将速度单位 km/h 转变为 mph，其值为 5/8。

(2) 求取滑艇速度控制系统在此工作点处的平衡状态。

在 MATLAB 命令窗口中使用 trim 命令获得系统在输出为 100 mph 时的平衡状态：

```
>> [x, u, y, dx]=trim('wherry_control', [ ], [ ], 100, [ ], [ ], 1)
```

```
x =
    160
u =
  2.5440e+004
y =
    100
dx =
 -1.0914e-014
```

其中 trim 命令用来求取滑艇速度控制系统在速度稳定在 100 mph 时的平衡点，x 表示滑艇速度控制系统中的状态变量，即子系统模块中积分模块的输出；u 表示滑艇在平衡点处的系统输入值，即滑艇牵引力；y 表示滑艇在平衡点处的系统输出，即滑艇速度。

(3) 求取滑艇速度控制系统的线性系统描述。

在获得使滑艇速度稳定在 100 mph 处时系统的平衡点 x、u 与 y 之后，在 MATLAB 命令窗口中使用 linmod 命令便可以获得相应的线性系统描述，如下所示：

```
>> [A,B,C,D]=linmod('wherry_control',x,u)
A =
    -0.3190
B =
  1.0000e-003
C =
    0.6250
D =
    0
```

从而得到线性化后系统的状态空间描述，其中 A、B、C 与 D 是线性系统的状态空间矩阵。故相应的线性系统的状态空间描述方程为

$$\dot{x} = -0.319x + 0.001u$$

$$y = 0.625x$$

注意：此状态空间描述是在系统平衡点 $x = 160$，$u = 25440$ 以及 $y = 100$ 处附近的近似表示。此时系统为稳定的系统，但是由于矩阵 A 的值为负值，故当系统在其它的工作点处可能不稳定。

4. 使用 LTI Viewer 进行非线性系统的线性分析

除了使用前面的命令行方式对非线性系统进行线性化处理分析之外，Simulink 还提供了友好的图形界面对非线性系统进行线性分析。使用图形界面可以使用户对非线性系统的性能有一个非常直观的认识与理解。用户可以从相应的系统输出图形中来定性判断系统输出是否满足设计要求。下面简单介绍 LTI Viewer（Linear Time-Invariant，线性时不变系统浏览器）图形界面的功能与使用并仍以滑艇速度控制系统 wherry_control 为例进行说明。

在滑艇控制系统模型 wherry_control 中，选择 Tools 菜单下的 Linear Analysis 命令。此时将打开 LTI 线性时不变系统浏览器窗口，如图 10.4 所示。

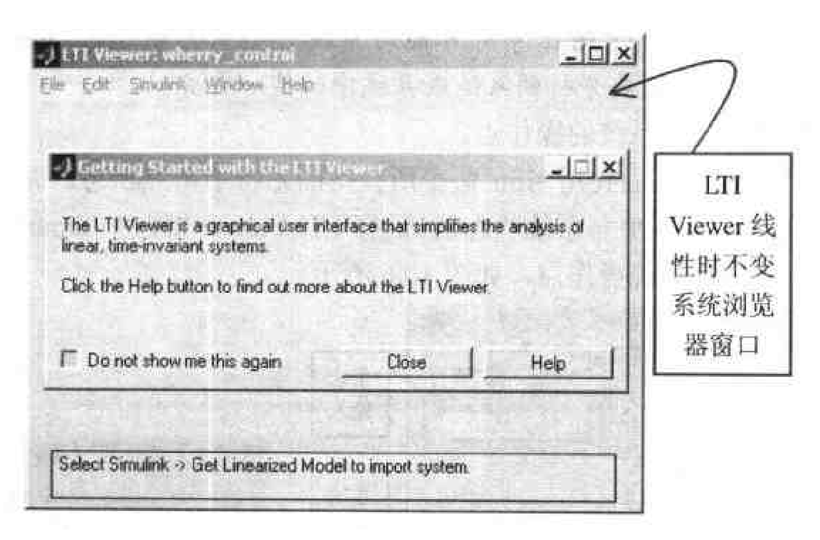


图 10.4　LTI Viewer 窗口界面

在对非线性系统使用 LTI Viewer 进行首次线性分析时，Simulink 会弹出一个简单介绍 LTI Viewer 的对话框，如图 10.4 所示。为对 LTI Viewer 作进一步的了解，用户可以单击 Help 按钮获得 LTI Viewer 的联机帮助或单击 Close 按钮关闭此对话框。与此同时还打开包含系统分析输入点（Input point）模块与系统分析输出点（Output point）模块在内的模块库 Model_Inputs_and_Outputs。在对非线性系统进行线性分析时，用户必须指定所分析的非线性系统中的参考输入点（即系统分析输入点 Input point）与参考输出点（即系统分析输出点 Output point）。对于 wherry_control 滑艇速度控制系统而言，使用 LTI Viewer 对其进行线性分析的步骤如下：

(1) 在滑艇速度控制系统模型中加入 Input point 与 Output point，如图 10.5 所示。

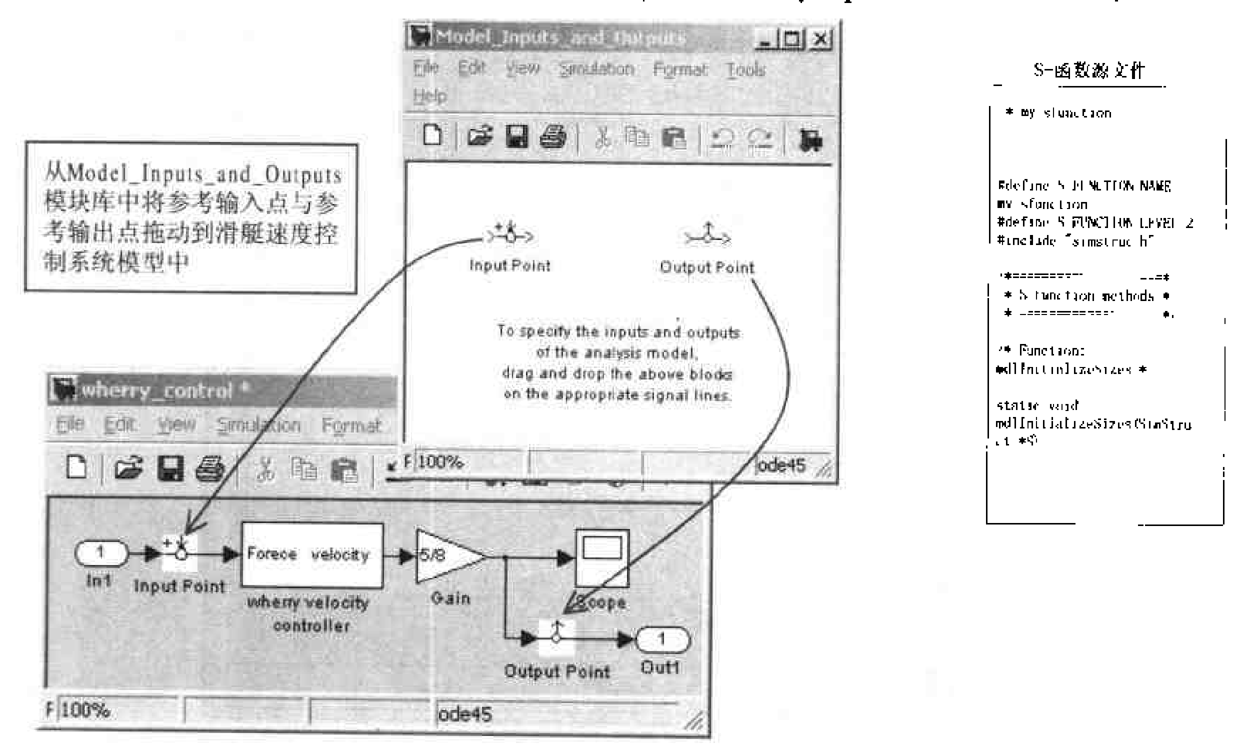


图 10.5　在系统模型中加入 Input point 与 Output point

注意：参考输入点与参考输出点有可能与系统模型的系统输入 Inport 与系统输出 Outport 不相同，这取决于所要分析的系统或系统中的某一部分。

(2) 设置滑艇速度控制系统的操作点。

所谓的操作点便是在前面使用 trim 命令所获得的系统平衡点状态，即使滑艇速度能够稳定在 100 mph 的系统输入值与系统状态值。选择 LTI Viewer 窗口中 Simulink 菜单下的 Set Operating Point 命令设置系统操作点，如图 10.6 所示。

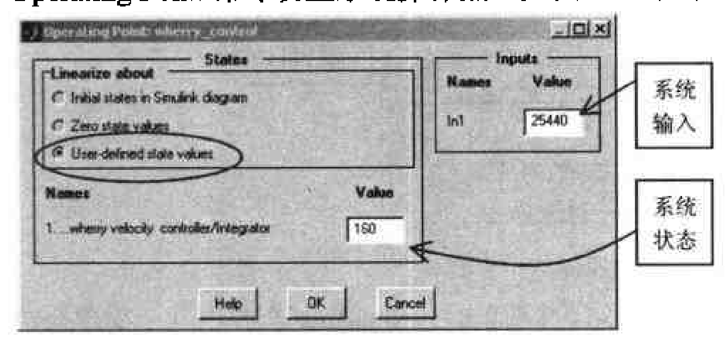


图 10.6 设置系统操作点

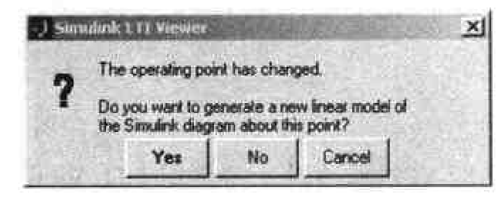


图 10.7 操作点改变提示对话框

注意：由于滑艇速度由用户来设置，因此在操作点设置中首先应该选择 User-defined states values，使用用户自定义状态值作为操作点状态值。在此，系统的操作点为 x=160，u = 25440，其设置如图 10.6 中所示。

(3) 获得滑艇速度控制系统在当前操作点的线性表示。

在对滑艇速度控制系统的操作点进行正确设置之后，单击 OK 按钮。此时将弹出如图 10.7 所示的对话框，以提示用户系统操作点的改变，选择 Yes 应用新的操作点设置。随后将在 LTI Viewer 窗口中显示滑艇速度控制系统的阶跃响应，如图 10.8 所示。

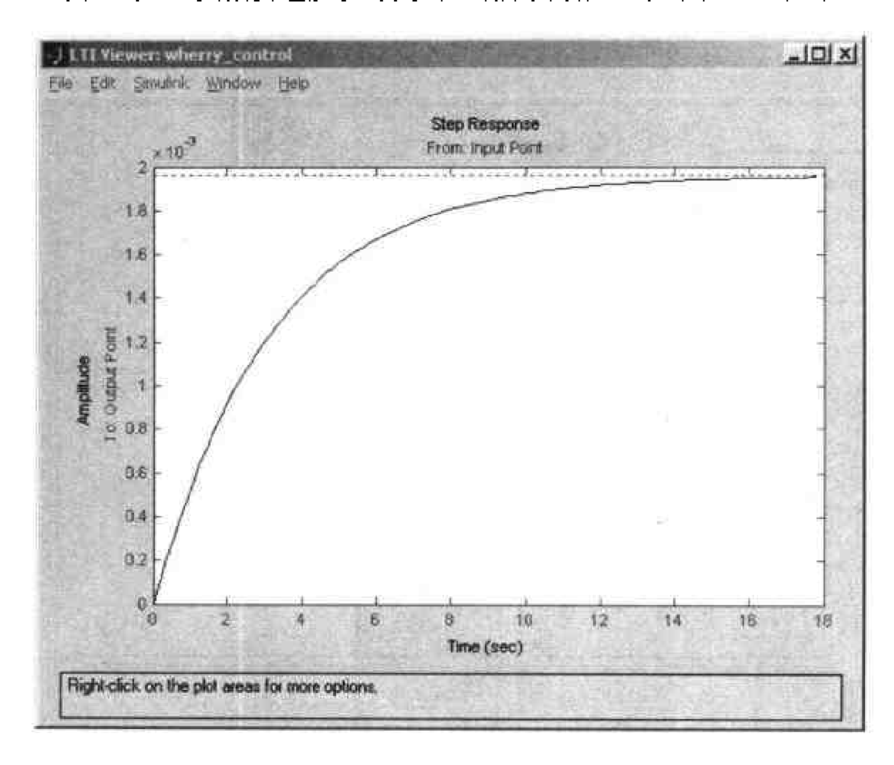


图 10.8 滑艇速度控制系统的阶跃响应

从滑艇速度控制系统的阶跃响应可以明显看出，此系统在操作点 x=160、u=25440 处为稳定系统。对于不同的操作点而言，系统的响应可能大不相同，甚至造成系统的不稳定。用户可以尝试不同的操作点设置，在 LTI Viewer 观测系统响应以对其进行分析，这里不再赘述。

10.1.2　线性时不变系统浏览器 LTI Viewer 介绍

在默认情况下，使用 LTI Viewer 进行系统的线性分析时，LTI Viewer 浏览器窗口所显示的图形为系统在单位阶跃信号作用下的系统响应。其实，LTI Viewer 浏览器提供了极其丰富的功能，它可以使用户对系统进行非常详细的线性分析。下面仍以滑艇速度控制系统 wherry_control 为例对 LTI Viewer 进行详细的介绍与说明。

1. 绘制系统的不同响应曲线

在默认的情况下，LTI Viewer 绘制系统在单位阶跃信号输入下的系统响应曲线（即阶跃响应）。其实使用 LTI Viewer 可以绘制不同的系统响应，在 LTI Viewer 图形绘制窗口中单击鼠标右键，选择弹出菜单 Plot Type 下的子菜单，可以在 LTI Viewer 图形绘制窗口中绘制不同的系统响应曲线，如图 10.9 所示。

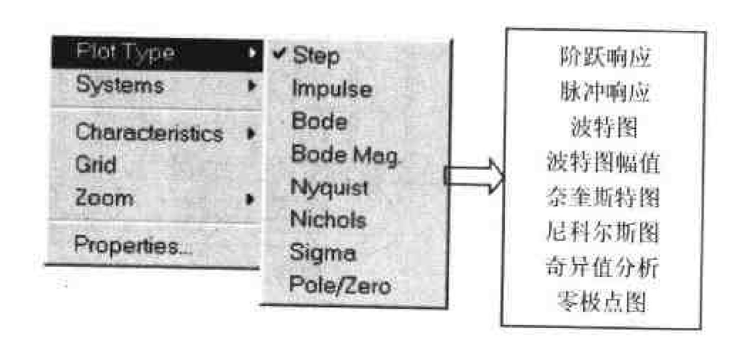


图 10.9　系统响应曲线绘制选择

如果用户选择 Impulse 命令，则可以绘制系统的单位脉冲响应曲线，如图 10.10 所示。

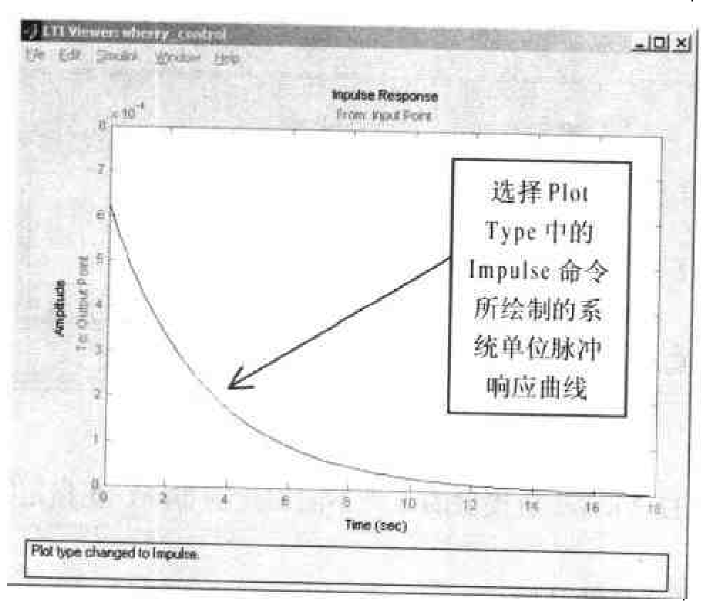


图 10.10　滑艇速度控制系统的单位脉冲响应曲线

除此之外，使用 LTI Viewer 还可以绘制系统的波特图（Bode）、波特图幅值图（Bode Mag）、奈奎斯特图（Nyquist）、尼科尔斯图（Nichols）、奇异值分析（Sigma）以及零极点图（Pole/Zero）等，其方法与绘制脉冲响应一致。

2. 改变系统响应曲线绘制布局

在默认的情况下，LTI Viewer 图形绘制窗口中仅仅绘制一个系统响应曲线。如果用户需

要同时绘制多个系统响应曲线图，则可以使用 LTI Viewer 窗口中 Edit 菜单下的 Plot configurations 对 LTI Viewer 图形绘制窗口的布局进行改变，并在指定的位置绘制指定的响应曲线。图 10.11 为响应曲线绘制布局设置对话框，以及采用图中给出的设置同时绘制 6 幅不同的响应曲线。

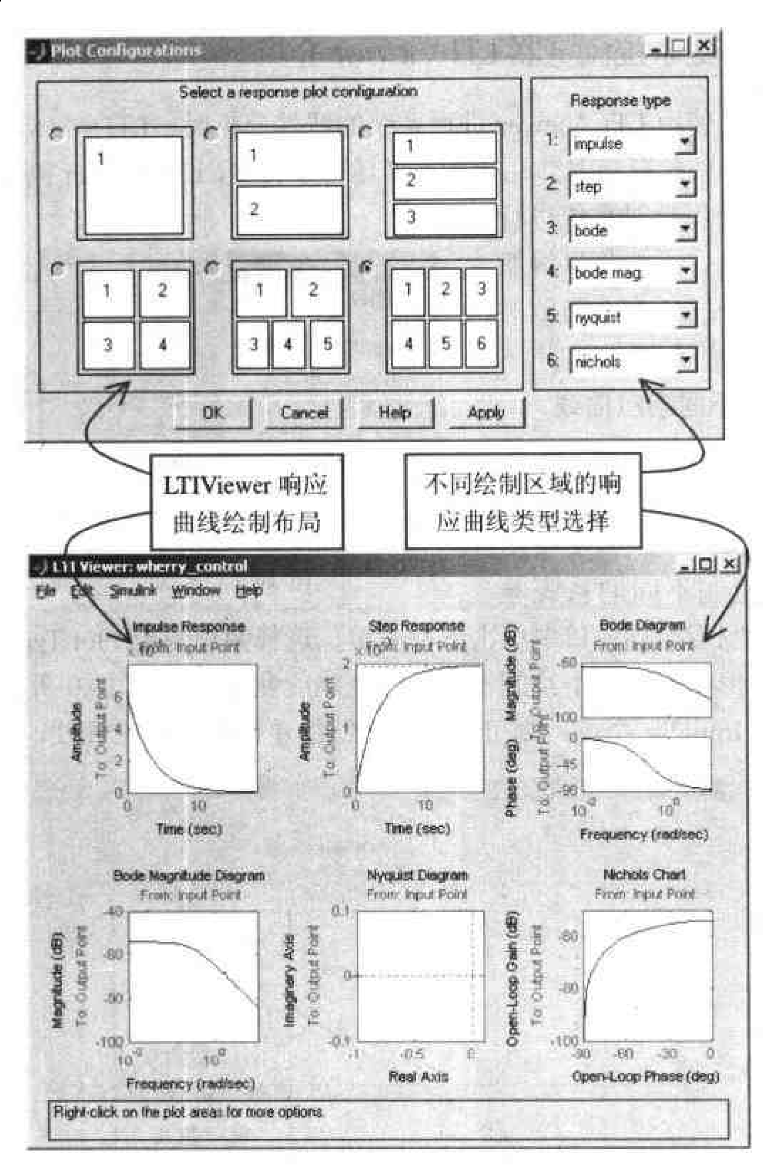


图 10.11 响应曲线布局设置及绘制结果

用户可以选择 LTI Viewer 所提供的 6 种不同的绘制布局，在指定的区域绘制自己感兴趣的响应曲线。

3. 系统时域与频域性能分析

使用 LTI Viewer 不仅可以方便地绘制系统的各种响应曲线，还可以从系统响应曲线中获得系统响应信息，从而使用户可以对系统性能进行快速的分析。首先，通过单击系统响应曲线上任意一点，可以获得动态系统在此时刻的所有信息，包括运行系统的名称、系统的输入输出以及其它与此响应类型相匹配的系统性能参数。例如，对于滑艇速度控制系统的单位脉冲响应，单击响应曲线中的任意一点，可以获得系统响应曲线上此点所对应的系统运行时刻（Time）、系统输入值（Amplitude）等信息，如图 10.12 所示。

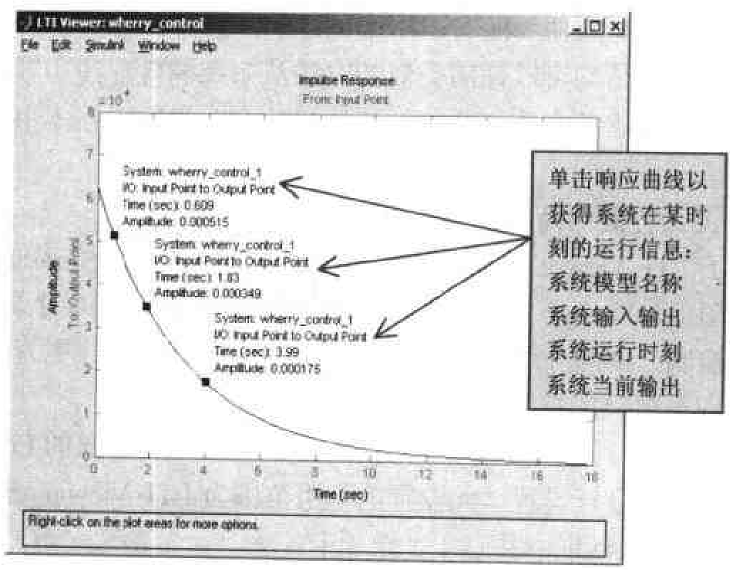


图 10.12　从系统响应曲线获得系统运行信息

其次，用户可以在 LTI Viewer 图形绘制窗口中单击鼠标右键，使用右键弹出菜单中的 Characteristics 子菜单获得系统不同响应的特性参数，对于不同的系统响应类型，Characteristics 菜单的内容并不相同。图 10.13 所示为阶跃响应的特性参数。

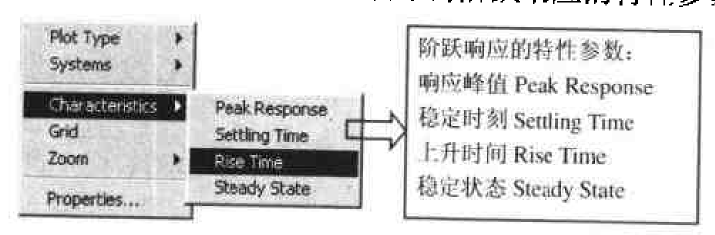


图 10.13　阶跃响应的特性参数

选择 Characteristics 右键弹出菜单中的 Rise Time 可以获得系统阶跃响应的上升时间（即系统输出从终值的 10%到终值的 90%所需要的时间）。此时在 LTI Viewer 绘制的阶跃响应曲线中将出现上升时间标记点，单击此标记点即可获得上升时间，如图 10.14 所示。

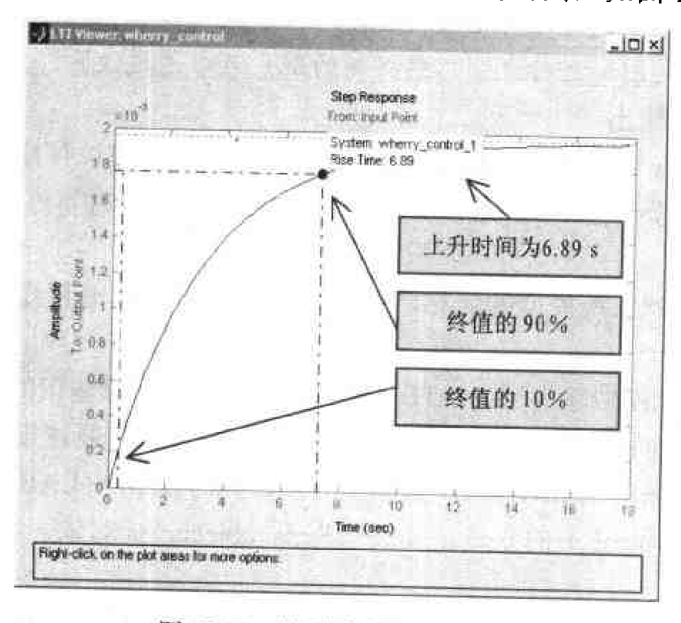


图 10.14　阶跃响应的上升时间

对于不同类型的系统响应曲线而言，用来描述响应特性的参数各异。虽然不同响应曲线的特性参数不相同，但是均可以使用类似的方法从系统响应曲线中获得相应的信息。用户可以尝试一下，至于描述各个系统响应的特性参数的含义则不在本书的内容之中，用户可以参考相关的书籍。

4. LTI Viewer 图形界面的高级控制

前面简单介绍了 LTI Viewer 响应曲线绘制窗口的布局设置。Simulink 最为突出的特点之一就是其强大的图形功能。在 Simulink 中，任何图形都是特定的对象，用户可以对其进行强有力的操作与控制。下面介绍如何对 LTI Viewer 图形窗口进行更为高级的控制。

对 LTI Viewer 图形窗口的控制有两种方式。

一是对整个浏览器窗口 Viewer 进行控制：单击 LTI Viewer 窗口的 Edit 菜单下的 Viewer Preferences 命令对浏览器进行设置（此设置的作用范围为 LTI Viewer 窗口以及所有系统响应曲线绘制区域）。在此对话框中共有 4 个选项卡：

(1) Units 选项卡：设置图形显示时频率、幅值以及相位的单位。

(2) Style 选项卡：设置图形显示时的字体、颜色以及绘图网格。

(3) Characteristics 选项卡：设置系统响应曲线的特性参数。

(4) Parameters 选项卡：设置系统响应输出的时间变量与频率变量。

二是对某一系统响应曲线绘制窗口进行操作：在系统响应曲线绘制窗口中单击鼠标右键，选择弹出菜单中的 Properties 对指定响应曲线的显示进行设置。此对话框中共有 5 个选项卡：

(1) Labels 选项卡：设置系统响应曲线图形窗口的坐标轴名称、窗口名称。

(2) Limits 选项卡：设置坐标轴的输出范围。

(3) Units 选项卡：设置系统响应曲线图形窗口的显示单位。

(4) Style 选项卡：设置系统响应曲线图形窗口的字体、颜色以及绘制网格。

(5) Characteristics 选项卡：设置系统响应曲线的特性参数。

注意：*对于不同的系统响应曲线，其特性参数不相同，故 Characteristics 选项卡中内容也不相同。由于使用如上的方法对 LTI Viewer 图形界面的控制非常简单而且直观，限于篇幅，这里不再以实际的图形进行说明。感兴趣的用户可以通过设置以获得不同的显示效果。*

5. 线性化模型的输出

在使用 LTI Viewer 对非线性系统进行线性分析之后，用户可以使用 File 菜单下的 Export 命令将此线性化线性模型输出到 MATLAB 工作空间之中。此时将打开如图 10.15 所示的窗口。

图中列出了一系列系统模型的名称，选择其中的一个以输出其线性化模型。输出方式有两种：

(1) 将系统线性化模型输出至 MATLAB 工作空间中，此时输出的线性化模型为一 LTI 对象变量（LTI 对象的相关知识将在下面的内容中给出）。将滑艇速度控制系统在给定的操作点处的线性化模型输出到 MATLAB 工作空间中，此时在 MATLAB 工作空间列表中将出现一个名为 wherry_control_1 的变量，其类型为 ss object（ss 对象，属于 LTI 对象），如图 10.16 所示。

(2) 将系统线性化模型输出至磁盘文件（*.mat 文件，即 MATLAB 数据文件）。

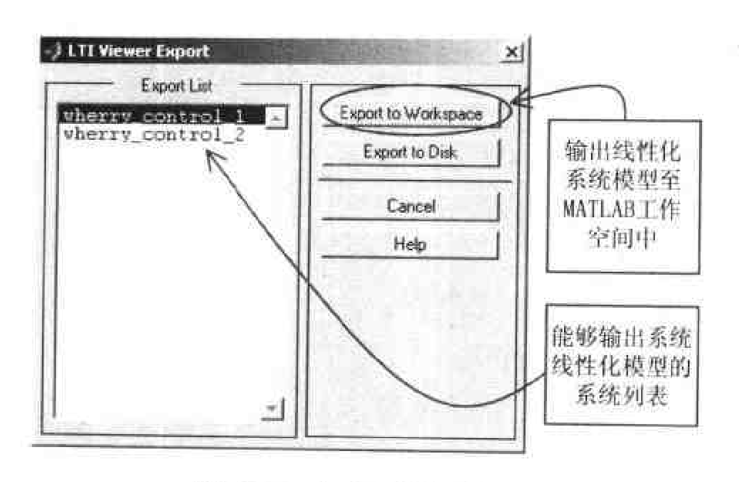


图 10.15　输出线性化系统模型

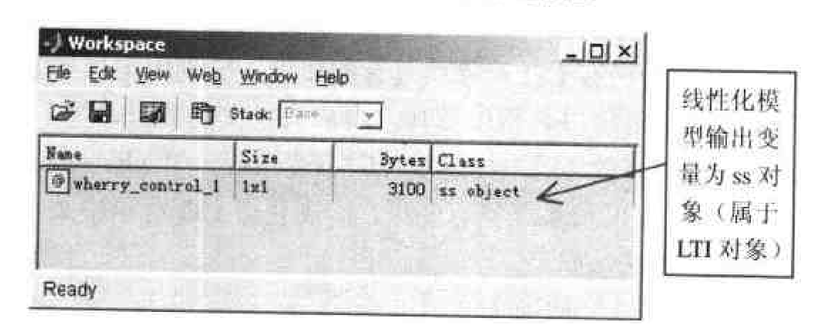


图 10.16　线性化系统模型的输出变量

10.1.3　LTI 线性时不变系统对象介绍

在使用 LTI Viewer 线性时不变系统浏览器对系统进行线性分析时，如果将线性化的系统模型输出至 MATLAB 的工作空间中，则会在 MATLAB 工作空间中出现类型为 ss 对象的变量（ss object）。ss 对象是 LTI 对象（线性时不变系统对象）的一种形式，LTI 对象是控制系统工具箱中最为基本的数据类型。此类型数据拥有描述一个线性时不变系统的所有信息，LTI 对象有如下的三种方式：

(1) ss 对象：封装了由状态空间模型描述的线性时不变系统的所有数据。

(2) tf 对象：封装了由传递函数模型描述的线性时不变系统的所有数据。

(3) zpk 对象：封装了由零极点模型描述的线性时不变系统的所有数据。

1. LTI 对象的属性

不同的 LTI 对象除了拥有某些共同的属性之外，还有属于每一种对象本身的特殊属性。使用 get 命令可以获得 LTI 对象的所有属性。对于由 LTI Viewer 输出的描述滑艇速度控制系统的线性化模型的 ss 对象 wherry_control_1 而言，使用 get 命令获得 ss 对象的属性如下：

```
>>get(wherry_control_1)
                    a: -0.319
                    b: 0.001
                    c: 0.625
                    d: 0
```

```
             e: []
     StateName: {[1x53 char]}
            Ts: 0
   ioDelay: 0
    InputDelay: 0
   OutputDelay: 0
    InputName: {'Input Point'}
   OutputName: {'Output Point'}
   InputGroup: {0x2 cell}
  OutputGroup: {0x2 cell}
        Notes: {}
     UserData: []
```

其中从 Ts 开始之后的属性为所有 LTI 对象均具有的属性，分别用来描述 LTI 系统的采样时间、输入输出延迟、输入输出端口名称以及其它用户自定义的数据等等。而在 Ts 之前的属性则属于不同对象本身所特有的，用来描述线性时不变系统，如 wherry_control_1 中的状态空间描述矩阵 a、b、c、d、e 以及系统状态变量。如需获得 LTI 对象中某个属性的取值，则可以使用与结构体类似的语法，如

```
>>state=wherry_control_1.StateName
state =
    'wherry_control/wherry velocity controller/Integrator'
```

相应地，使用 set 命令可以对 LTI 对象的指定属性进行修改，其使用方法与设置系统模型或其中的系统模块的属性相类似。

2. 对 LTI 对象的基本操作

由于 LTI 对象是控制工具箱中最基本的数据类型，因而 MATLAB 支持对 LTI 对象的直接操作。用户可以使用控制工具箱中的系统分析设计命令对这些 LTI 对象进行操作，而且由于 LTI 对象包括线性系统是连续还是离散的信息，因此可以使用同样的命令对连续系统与离散系统进行操作。这里并不打算对控制工具箱中的函数命令进行介绍，而仅介绍 LTI 对象本身的一些简单操作。

(1) 生成 LTI 对象。使用 ss、tf 及 tpk 可以建立不同类型的 LTI 对象，如使用 tf 命令建立使用传递函数描述的线性时不变系统对象。

```
>>mysys_tf=tf([2 1], [1 3 2])          %生成 tf 对象 mysys_tf
Transfer function:
   2 s + 1
-------------------
s^2 + 3 s + 2
```

(2) LTI 对象间的相互转换。同样可以使用 ss、tf 及 zpk 进行 LTI 对象之间的相互转换，如

```
>>mysys_ss=ss(mysys_tf)        %将 tf 对象转换为 ss 对象
a =
```

```
      x1   x2
   x1   -3   -1
   x2    2    0
b =
      u1
   x1    2
   x2    0
c =
      x1     x2
   y1      1   0.25
d =
      u1
   y1    0
Continuous-time model.              % 指明系统为连续时间系统
```

(3) 获得与设置 LTI 对象属性。在前面已经介绍，这里不再赘述。

(4) 线性时不变系统的并联，即 LTI 对象的相加，如

```
>>sys1=tf([2 1],[1 3 2]);                  % 生成系统 1
>>sys2=tf([1 1],[ 2 3 -1]);                % 生成系统 2
>>sys=sys1+sys2                            % 并联系统 1 与 2
Transfer function:
     5 s^3 + 12 s^2 + 6 s + 1
-------------------------------------------
2 s^4 + 9 s^3 + 12 s^2 + 3 s - 2
```

本节详细介绍了如何对非线性系统进行线性分析。在使用 Simulink 设计与分析动态系统时，线性分析是最为常用的技术之一，这是因为对非线性系统的设计与实现都远远难于线性系统。因此，希望用户能够很好的理解本节的内容，下面的两节内容将对使用控制工具箱设计控制系统做一个简单的介绍。

10.2 线性控制系统设计分析

在控制系统的设计分析之中，线性系统的设计、仿真分析与实现具有重要的地位。在 MATLAB 中所提供的控制系统工具箱对控制系统的设计提供了强大的支持，用户可以使用控制系统工具箱设计与分析控制系统；然后使用 Simulink 对所设计的控制系统进行仿真分析，并在需要的情况下修改控制系统的设计以达到特定的目的，从而使得用户快速完成系统设计的任务，大大提高设计的效率。本节将对使用控制系统工具箱进行控制系统设计，以及使用 Simulink 对设计好的系统进行验证进行简单的介绍。

10.2.1 控制系统工具箱简介

控制系统工具箱是MATLAB中所提供的对控制系统进行辅助设计的功能强大的开发设计工具。它包含了丰富的线性系统分析和设计函数，并以 LTI 对象为基本数据类型对线性时不变系统进行操作与控制。控制系统工具箱能够完成系统的时域和频域分析。在控制系统工具箱中，可以使用不同的方法设计线性反馈系统，如

(1) 根轨迹设计分析法。

(2) 极点配置法。

(3) H_2 和 H_∞ 控制。

(4) 状态观测器设计。

(5) 规范型实现设计。

在使用控制系统工具箱完成线性反馈系统设计之后，便可以通过 Simulink 进行系统的动态仿真，从而得到真实的、非线性系统的响应，进一步对控制器进行验证。

10.2.2 系统分析与设计简介

控制系统工具箱中最基本的数据类型为 LTI 对象。无论 LTI 对象的类型如何，都可以使用相同的命令对其进行分析，因为 LTI 对象包含了线性时不变系统的所有信息。这里简单介绍一下用来对由 LTI 对象所描述的线性时不变系统进行分析设计的命令函数。

1. 动态分析函数

动态分析函数有 pole(sys)、dcgain(sys)、tzero(sys)、damp(sys)及 norm(sys)等等。对于由如下命令：

```
>>mysys_tf=tf([2 1],[1 3 2]);
```

生成的 LTI 对象 mysys_tf 所描述的线性时不变系统，可以使用上述函数对其进行分析，例如：

```
>>pole (mysys_tf)            % 求取系统极点
ans =
    -2
    -1
>> dcgain(mysys_tf)          % 求取系统直流增益
ans =
0.5000
```

2. 时域与频域分析函数

时域与频域分析函数有 step(sys)、bode(sys)、impulse(sys)、nichols(sys)、initial(sys,x0)、nyquist(sys)、lsim(sys,u,t)以及 sigma(sys)等。例如：

```
>> step(mysys_tf)            % 绘制系统的单位阶跃响应曲线
>> figure, nyquist(mysys_tf) % 在新的图形窗口绘制系统的 nyquist 图
```

使用这两个命令分别绘制线性时不变系统 mysys_tf 的单位阶跃响应与 nyquist 图，与对 LTI Viewer 中系统响应曲线的操作相类似，用户可以使用右键弹出式菜单获得系统的时域（或频域）的动态响应（或动态性能），如图 10.17 所示。

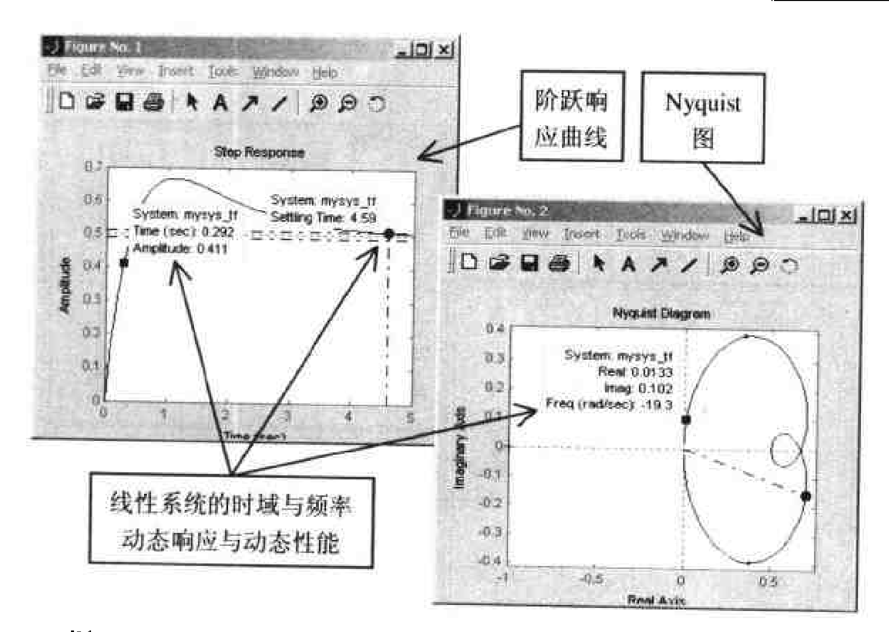


图 10.17　线性时不变系统 mysys_tf 的阶跃响应曲线与 Nyquist 图

3. 补偿器设计

使用控制系统工具箱中的函数还可以进行各种系统的补偿设计，如 LQG（Linear-Quadratic-Gaussian，线性二次型设计）、Root Locus（线性系统的根轨迹设计）、Pole placement（线性系统的极点配置）以及 Observer-based regulator（线性系统观测器设计）等。由于这些内容涉及较多的知识，在此不作介绍。

在实际的系统设计中，只要系统经过线性化处理，使用 LTI 线性时不变系统模型来表示，用户都可以使用若干个线性系统控制器的设计方法来进行设计。

10.2.3　单输入单输出系统设计工具

在对非线性系统的线性分析技术进行介绍时，线性时不变系统浏览器 LTI Viewer 是进行系统线性分析的最为直观的图形界面，使用 LTI Viewer 使得用户对系统的线性分析变得简单而直观。其实 LTI Viewer 只是控制系统工具箱中所提供的较为简单的工具，主要用来完成系统的分析与线性化处理，而并非系统设计。

SISO 设计器是控制系统工具箱所提供的一个非常强大的单输入单输出线性系统设计器，它为用户设计单输入单输出线性控制系统提供了非常友好的图形界面。在 SISO 设计器中，用户可以同时使用根轨迹图与波特图，通过修改线性系统零点、极点以及增益等传统设计方法进行 SISO 线性系统设计。下面仍以 tf 对象 mysys_tf 为例说明 SISO 设计器的使用。

1. 启动 SISO 设计器

在 MATLAB 命令窗口中键入如下的命令启动 SISO 设计器：

```
>>sisotool
```

启动后的 SISO 设计器如图 10.18 所示。

在默认的情况下，SISO 设计器同时启用系统根轨迹编辑器与开环波特图编辑器，如图 10.18 所示。当然，此时尚未进行系统设计，故不显示根轨迹与开环波特图。

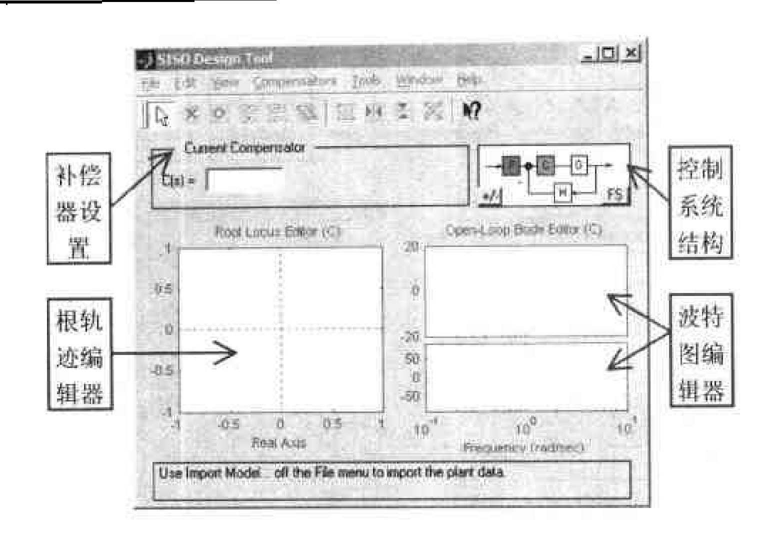


图 10.18 SISO 设计器

2. 输入系统数据（Import System Data）

在启动 SISO 设计器之后，需要为所设计的线性系统输入数据，选择 SISO 设计器中 File 菜单下的 Import 命令输入系统数据，此时将打开如图 10.19 所示的对话框。

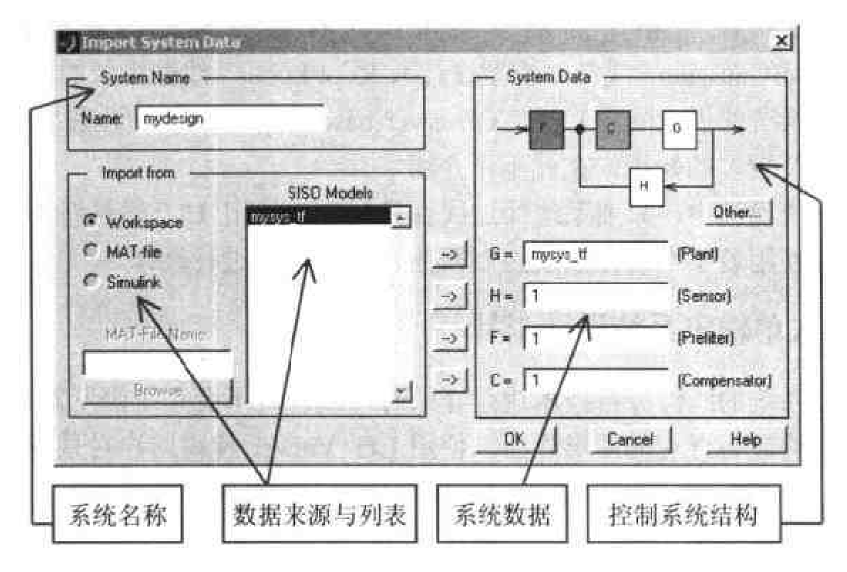


图 10.19 系统数据输入对话框

使用此对话框可以完成线性系统的数据输入。注意，如果数据来源为 Simulink 系统模型框图，则必须对其进行线性化处理以获得系统的 LTI 对象描述。这是因为 SISO 线性系统中的所有对象（G 执行结构、H 传感器、F 预滤波器、C 补偿器）均为 LTI 对象。另外，用户可以单击控制系统结构右下方的 Other 按钮以改变控制系统结构。改变后的控制系统结构示意图如图 10.20 所示。

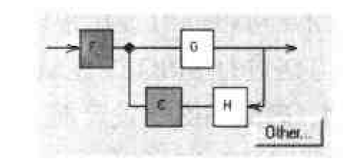


图 10.20 控制系统结构改变示意图

使用 SISO 默认的控制系统结构，并设置控制系统的执行结构（即控制对象）数据 G 为 mysys_tf，其它的参数 H、F、C 均使用默认的取值（常数 1）。然后单击 OK 按钮，此时在 SISO 设计器中会自动绘制此负反馈线性系统的根轨迹图以及系统开环波特图，如图 10.21

所示。

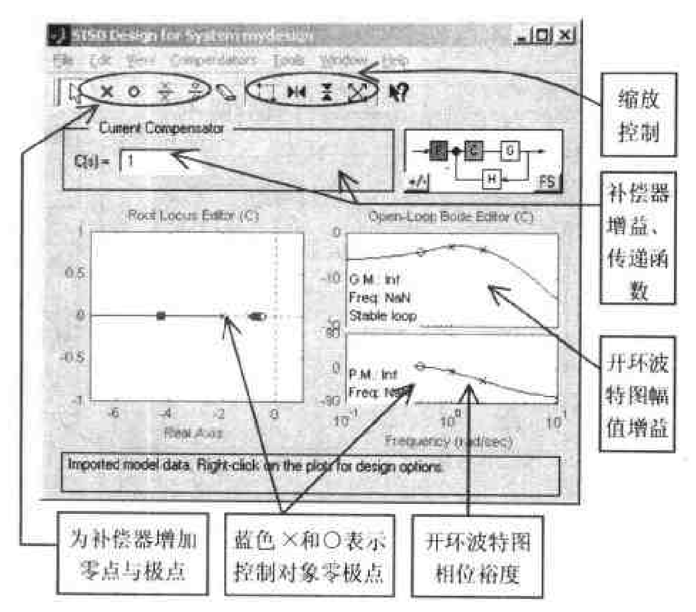


图 10.21　系统数据输入后的 SISO 设计器界面

说明：在系统的根轨迹图中，蓝色×和○表示控制对象 G 的零极点，而红色表示系统补偿器 C 的零极点。用户可以在根轨迹编辑器中对系统的根轨迹进行控制与操作：增加补偿器的零极点、移动零极点改变其分布、移动根轨迹图中的紫色方块改变系统增益等等，这些操作均可以改变系统的动态性能。另外，在波特图中除了显示当前补偿器下的系统增益与相位裕度之外，还显示了零点与极点的位置。

3. 设计与分析系统

在完成线性系统数据的输入之后，用户便可以使用诸如零极点配置、根轨迹分析以及系统波特图分析等传统的方法对线性系统进行设计。除了前面介绍的对系统零极点的各种操作（增加、删除以及改变分布）之外，SISO 中对线性系统的设计提供了诸多的支持，如：单击补偿器增益及传递函数区域可以弹出补偿器设置对话框，使用此对话框可以设置补偿器 C 的增益、零点及极点等，如图 10.22 所示。

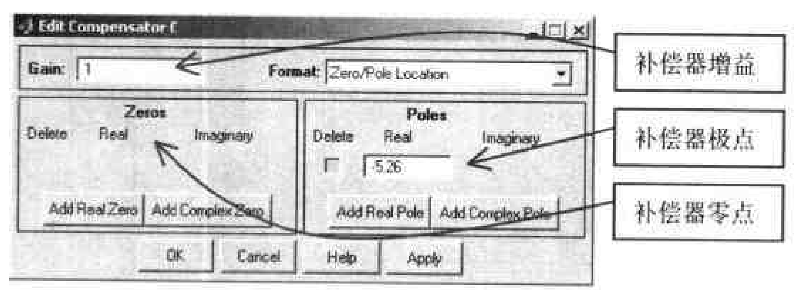


图 10.22　补偿器 C 的增益、零点及极点设置

限于篇幅，在此对 SISO 的操作不做过多的介绍，而主要介绍使用 SISO 设计器设计线性控制系统的思路与流程，以使用户能够对其有一个全面的认识，并且理解 Simulink 在实际系统设计分析中的应用。因此，在此系统设计中，我们仅仅为补偿器增加一个极点，如图 10.23 所示。

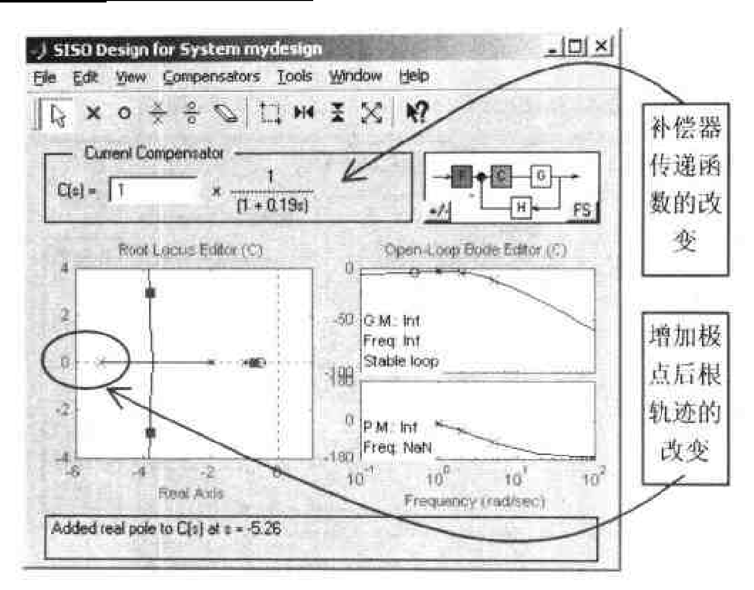


图 10.23　为系统增加极点

从系统的根轨迹图与系统波特图可以明显看出增加补偿器极点的影响。当然，任何的设计都不是随心所欲的，都必须按照指定的性能参数以及控制系统本身的规律进行设计，这里仅仅举例说明其设计的方法而已。

在系统设计完成后，需要对其做进一步的分析：分析反馈系统的开环和闭环响应，以确保系统是否满足特定的设计要求。用户可以选择 SISO 设计器中 Tools 菜单下的 Loop Responses 绘制指定的开环响应（或闭环响应）曲线。此时将打开 LTI 浏览器，用户可在 LTI 浏览器中对系统的性能如过渡时间、峰值响应、上升时间等等进行分析，如图 10.24 所示。

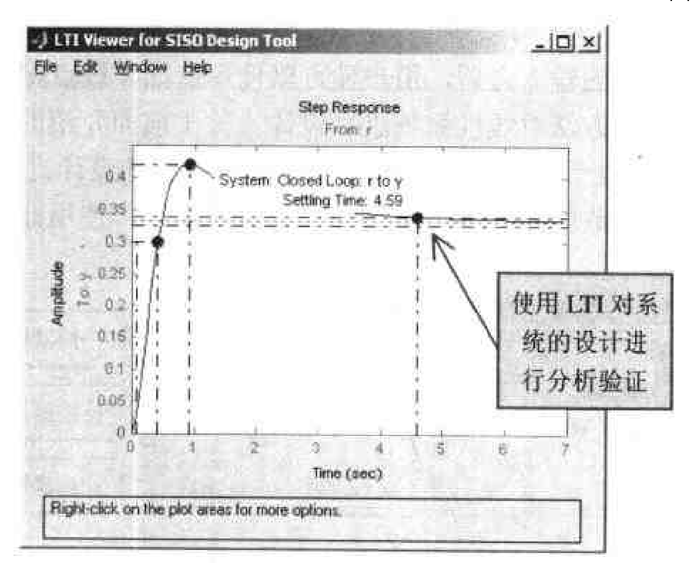


图 10.24　使用 LTI 对系统的设计进行分析验证

如果用户需要设计线性离散控制系统，可以选择 Tools 菜单下的 Continuous / Discrete Conversions 选项，以对离散控制系统的采样时间、连续信号的离散化方法等进行设置，如图 10.25 所示。

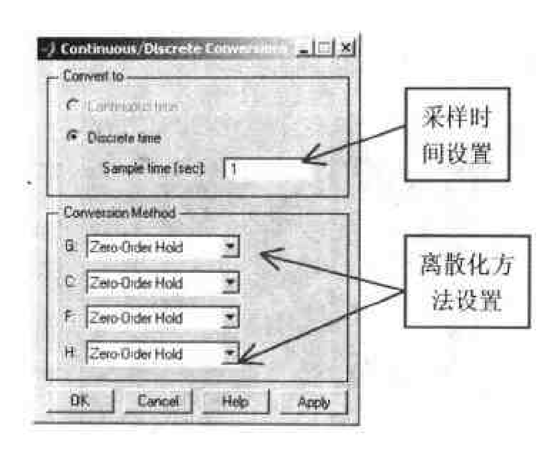


图 10.25　离散控制系统设计设置：采样时间与离散化方法

4. SISO 设计器与 Simulink 的集成：系统验证

在使用SISO完成系统的设计之后，在系统实现之前必须对设计好的系统进行仿真分析，以确保系统设计的正确性。如果直接按照系统设计逐步建立系统的 Simulink，将是一件很麻烦的工作；庆幸的是，SISO 提供了与 Simulink 集成的方法，用户可以直接使用 SISO 设计器 Tools 菜单下的 Draw Simulink Diagram 直接由设计好的系统生成相应的 Simulink 系统框图。在生成 Simulink 系统模型之前，必须保存线性系统的执行结构、补偿器以及传感器等 LTI 对象至 MATLAB 工作空间中。图 10.26 所示为此系统相应的 Simulink 系统模型以及 MATLAB 工作空间变量列表。

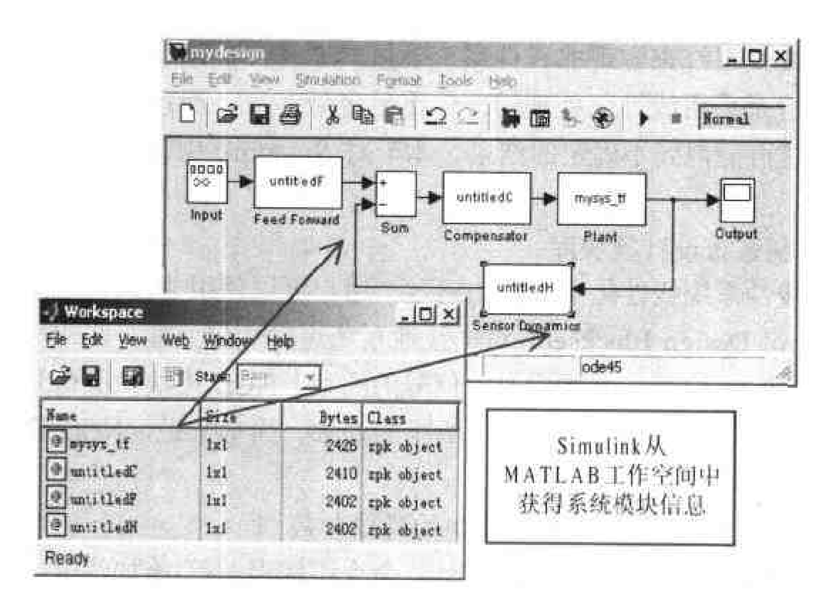


图 10.26　由设计好的系统直接生成相应的 Simulink 模型

注意：生成的 Simulink 系统模型的实现均采用了 MATLAB 工作空间中的 LTI 模块。在生成 Simulink 系统模型之后，便可以对设计好的系统进行仿真分析以验证系统设计的正确性。使用 Sources 模块库中的 Step 模块为系统提供单位阶跃输入信号（设置阶跃时刻为 0），然后运行仿真，图 10.27 为系统的仿真结果。系统的仿真结果与 LTI Viewer 中的阶跃响应曲线完全一致，从而验证了系统设计的正确性。

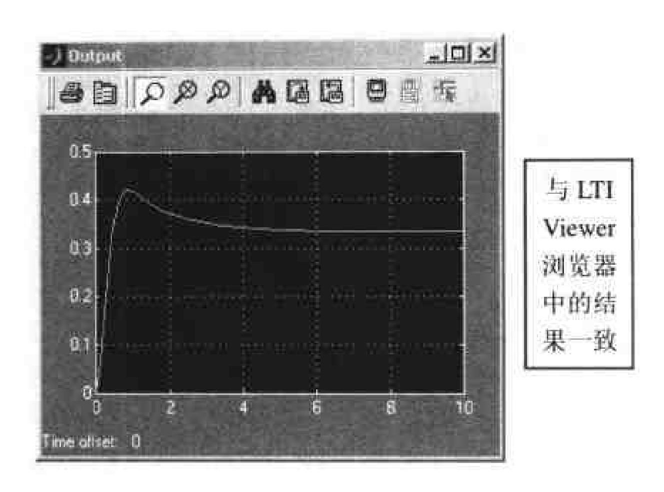


图 10.27 Simulink 系统模型的仿真结果

10.3 非线性控制系统设计简介

10.2 节对线性控制系统的设计与分析做了比较详细的介绍。由于线性控制系统的设计算法已经相当成熟，设计者可以非常容易的使用线性控制工具箱中的诸多工具来设计线性控制系统。然而，对于非线性控制系统的设计算法，还很不成熟。因此，在对非线性控制系统进行设计分析时，总是通过在一定的系统操作点处进行线性化分析，在距离操作点很小的范围内使用线性系统来实现非线性系统。虽然非线性控制系统的线性化设计思路得到了广泛的应用，但是采用线性化处理的一个不足之处在于：当系统的工作点远离系统指定的操作点时，系统可能得到不确定的结果。本节对 Simulink 中的非线性控制系统工具箱进行简单的概述。

1. 非线性控制系统设计模块库

为了解决非线性系统线性化中存在的问题，用户可以使用非线性控制设计模块库 NCD（Nonlinear Control Design Blockset）对非线性系统进行优化处理。用户可在指定的信号上连接一个 NCD Outport 模块，并确定对此信号的约束。NCD 模块按照信号的约束优化非线性系统中控制器的参数，使系统能够满足约束的要求。简单来说，NCD 模块可以为系统设计人员提供如下的功能：

(1) 受时域约束和不确定性作用下模块的最佳参数。

(2) 对任何信号(驱动信号、传感器信号、命令跟踪误差等)施加约束。

(3) 在优化过程中可以随时停止，并查看中间结果。

2. 非线性系控制系统设计

在使用 Nonlinear Control Design Blockset 之前，需要在 Simulink 中建立非线性控制系统（闭环形式）的系统模型。对于系统的各种模块（如控制器、补偿器等），无论是线性的还是非线性的，都应该在 MATLAB 的工作空间中定义这些模块的初始值信息。NCD Outport 模块必须连接到待控制的信号上。然后在这些模块上定义约束条件，这里的约束条件是针对连接点处的真实信号而言的，通常是一定幅值下的阶跃形式。在对非线性控制系统的设计完成之后，便可以使用 NCD Outport 模块对系统中各模块的参数进行优化，以使系统的

性能能够满足给定的约束条件。

3. 约束窗口

在建立好非线性控制系统模型并运行系统仿真之后，双击 NCD Outport 以打开约束窗口。在约束窗口中显示三个图形：约束条件（由红色条显示）、系统在原来参数下的响应曲线（由白色曲线显示）以及中间响应（由绿色曲线显示）。如果系统的响应不满足约束（比如超出了黑色区域），优化继续进行。并且用户可以在任何时候停止优化并检查结果。至于对 NCD Outport 约束窗口的诸多操作与控制，这里不再介绍。最后，在关闭 NCD Outport 约束窗口时，将会出现一个对话框以提示用户保存优化后系统模型中的模块参数（保存到 MATLAB 工作空间中）。图 10.28 所示为某非线性控制系统的约束窗口。

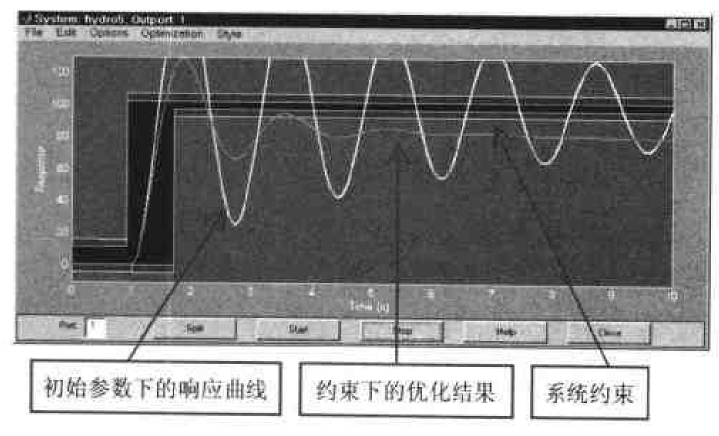


图 10.28　非线性控制系统的约束窗口

说明：在此约束窗口中初始参数下的响应曲线，约束下的优化结果曲线均是在单位阶跃信号输入下的响应，且阶跃时刻为 1。本节对非线性控制系统设计的介绍非常简单，其目的主要是使用户对其有一个感性的认识。至于控制系统的线性分析以及非线性控制系统设计的更多知识，则是在本书内容的范围之外。用户可以参考专业的技术资料对系统设计进行更多的认识与理解。



习　题

设非线性系统由下面的数学方程所描述：

$$\begin{cases} \ddot{x} = -0.5\dot{x} - 2x - x^2 + u \\ y = x \end{cases}$$

其中 u 为系统的输入，y 为系统的输出，x 为系统的状态变量。试对此非线性系统进行分析。要求如下：

(1) 建立系统的 Simulink 框图。

(2) 设系统输入信号为单位阶跃信号，观测此系统在初始条件 $x(0)=0, x(0)=0$ 及 $x(0)=0, x(0)=3$ 下的稳定性。

(3) 求取系统在输出 $y=20$ 时的平衡点，并在平衡点处的操作点附近对此系统进行线性化处理（求取系统的状态方程描述）。

(4) 使用 LTI Viewer 在与平衡点相应的操作点下对此系统进行线性分析，求取系统阶跃响应的特征参数，如上升时间、调节时间等。

(5) 输出线性化模型到 MATLAB 工作空间之中，并使用 SISO 设计器增加系统的极点（在 s 平面的左侧）以观测系统响应曲线的变化。

(6) 由 SISO 设计器所设计好的系统生成相应的 Simulink 系统模型框图，对系统进行仿真与验证。

提示：参考滑艇速度控制系统的线性化分析。

第 11 章　DSP Blockset

内容概要

- 基于帧的信号处理
- 滤波器设计
- 功率谱估计
- 矩阵操作

Simulink 环境是为动态系统仿真而设计的，它提供了建模和仿真的一些基本的算法和手段。而 Simulink 的 DSP Blockset 库为数字信号处理系统设计提供了丰富的模块，它包含基本的输入输出模块，以及其它操作处理模块如多采样率、自适应滤波、矩阵操作、统计、时频域转换等模块，这些模块之中包含了许多数字信号处理的核心算法。它们和 Simulink 是融为一体的，可以方便地和其它模块混合使用。DSP Blockset 不仅提供了丰富的数字信号处理模块，以使用户完成复杂的数字信号处理系统设计与仿真，而且还可以结合 RTW（Real Time Workshop）的强大功能，产生可以直接在数字信号处理硬件上运行的代码，这些代码对于对实时性要求不高的场合已经足够了。本章将简单介绍使用 DSP Blockset 进行数字信号处理系统设计的一些基础知识和基本技巧。

11.1　DSP 处理单元：帧

11.1.1　基于帧的信号处理

大多数实时的数字信号处理系统都采用基于帧的处理方式，以提高系统性能，这里每帧包含相邻的多个或者一组信号采样。采用基于帧的处理方式更适合多数的数字信号处理算法，另外也可降低系统对数据采集硬件的要求。缺省情况下，Simulink 所有信号都是基于采样的。这意味着在每个时间步，数据都被作为一个完整的采样进行处理，而不管它的维数大小。于是，仿真步长与一个采样周期相对应。然而在基于帧的处理中，包含若干采样的一帧数据在同一个时刻被处理。在这种情况下，一个仿真步长与帧周期相对应，即与一帧中包含的采样数与采样周期的乘积相对应。自然地，当需要考虑基于帧的信号的动态特性时，要特别小心，不要把采样周期、帧周期和仿真步长相混淆。帧可以用矩阵来表示，DSP Blockset 充分地利用了 Simulink 的矩阵功能来处理基于帧的信号。表 11.1 列出了基于采样的信号和基于帧的信号的对比关系。

表 11.1 基于采样的信号和基于帧的信号

基于采样的信号	基于帧的信号
每个时间步处理一个采样点	每个时间步处理含有 N 个采样点的一帧
仿真步长 = 采样周期 = T_s	仿真步长 = 帧周期 = $N*T_s$
采样频率 $F_s = 1/T_s$	帧频率 = F_s/N
采样步长可变	帧的大小可变，可以是工作区中的一个变量

之所以采用基于帧的处理主要是考虑到数字信号处理本身的要求和数据通讯的开销。显然，基于帧的信号处理应当比基于采样的处理要复杂得多，但是 Simulink 利用 MATLAB 的矩阵功能极大地提高了处理的效率。通过基于帧的处理，减少了块与块之间的通讯，从而比使用基于采样的信号进行仿真快得多。总之，利用基于帧的信号提高了仿真速度。而且，由于同样的原因，大多数 DSP 系统也采用基于帧的处理。除此之外，基于帧的处理提供了在仿真中进行频域分析的能力。

Simulink 的所有模块都支持基于帧的处理，使得用户可以方便地采用基于帧的信号进行算法仿真以及结合 RTW 产生实时代码。

图 11.1 说明了从连续信号经过 AD 采样得到采样信号，然后将采样信号组织成帧，送往 Simulink 处理的过程。

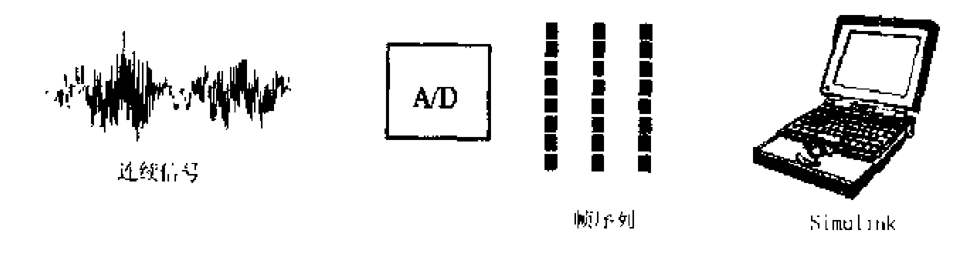


图 11.1 基于帧的信号处理

下面介绍帧信号在 Simulink 中的生成、显示等方面的内容。

1. 缓冲和解缓冲

在 Simulink 中采样信号和帧信号之间的转换是通过缓冲模块（Buffer）来实现的。Buffer 块有两种用途：一是接受采样输入并产生一定帧大小的帧输入；二是接受帧输入，修改帧的大小，这种情况下必须使用缓冲模块。这两种情况下都涉及到帧之间的重叠和帧的初始值的设置问题。当通过采样产生帧时，缓冲使用输入标量生成一个列向量，如图 11.2 所示。如果需要从一个帧信号产生一个采样信号，则应使用 Unbuffer 模块。

Source 库中的许多信号源模块同样提供基于帧的输出，当然使用这些模块作为输入信号时，就无需使用 Buffer 块，只需设置块的帧长参数就可以了。

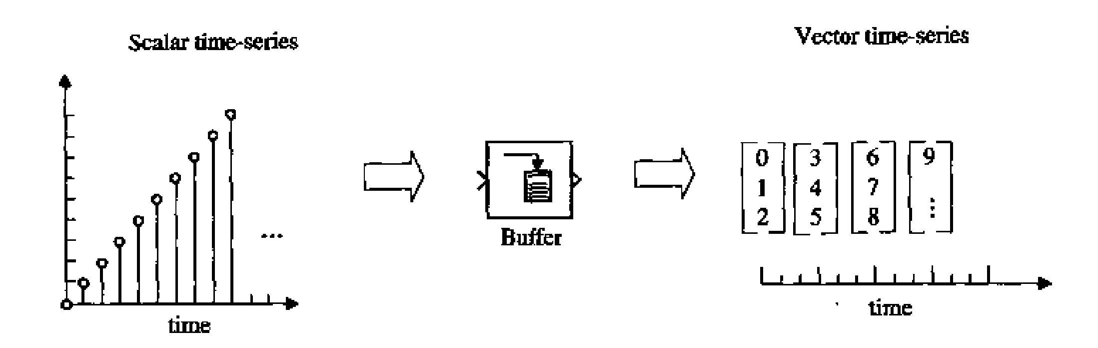
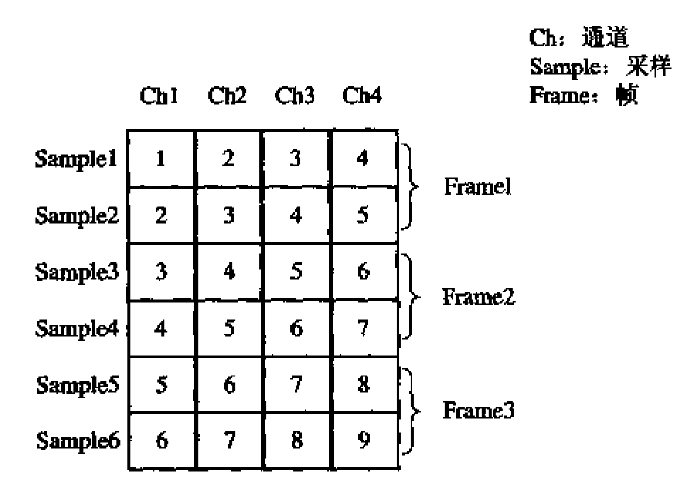


图 11.2　缓冲模块

2. 帧的表示

通常，一帧是通过一个矩阵表示的。在帧矩阵中，每个通道的信号对应矩阵中的一列，每个采样对应其中的一行（如图 11.3 所示）。在基于帧的处理中，各个模块沿着输入的每一列（通道）进行运算。图 11.3 中有四个信号通道，每帧有两个采样，帧和帧之间没有重叠。通常每帧的采样数是 2 的幂次，以满足 FFT 变换的需要。



	Ch1	Ch2	Ch3	Ch4
Sample1	1	2	3	4
Sample2	2	3	4	5
Sample3	3	4	5	6
Sample4	4	5	6	7
Sample5	5	6	7	8
Sample6	6	7	8	9

图 11.3　帧矩阵

3. 生成基于帧的信号

主要有三种方法用来生成基于帧的多通道信号。

(1) DSP 模块库中信号源库 DSP Sources 中的块提供了信号源块，用于生成基于帧的信号。在这些信号源中可以定义多个参数来生成多通道信号，如 DSP constant 块提供基于帧的或者基于采样的恒值输出，当使用基于帧的恒值输出时，Frame-based output 复选框必须选择，而且要提供帧的尺寸。如果帧的尺寸为 16 并确定通道数为 3(输入包含三个元素的行向量作为信号的参数)，那么这个块就会输出 16 乘 3 的矩阵（如图 11.4 所示）。

(2) 所有的信号都可以通过缓冲块成为帧。

(3) 将从若干个基于帧的信号源来的信号通过矩阵拼接成一个帧矩阵，形成一个多路信号。使用 \DSP Blockset\math functions\matrics & linear algbra\Matrix operations\ Matrix Concatenation 块将基于帧的单通道信号拼接起来时，注意 Concatenation Method （拼接方法）应设置为 Vertical（垂直拼接）。

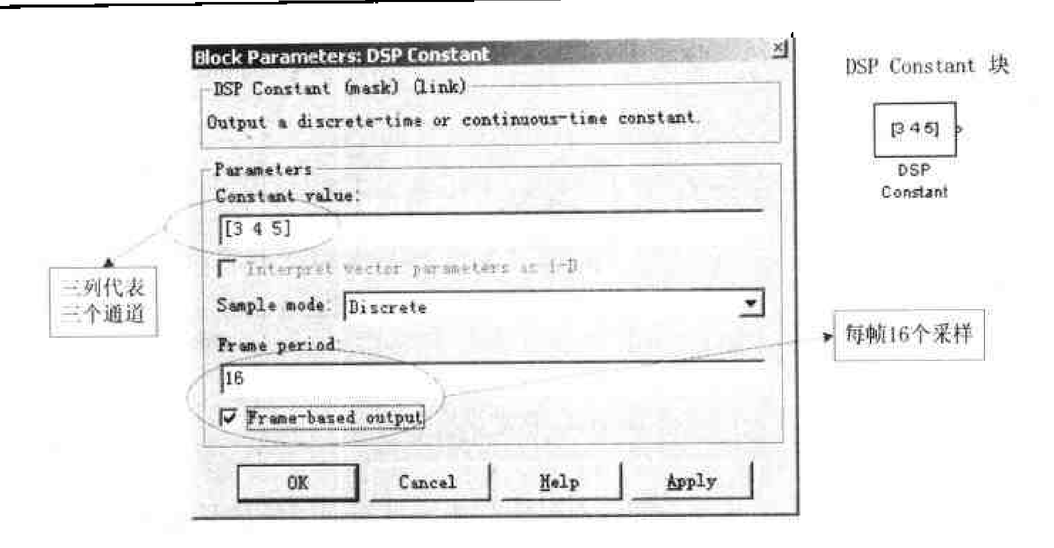


图 11.4 DSP Constant 模块设置

4. 观察基于帧的信号

用户可以使用 DSP blockset 提供的专门的显示模块来观察基于帧的信号。这些模块中最常用的是 Matrix Viewer（矩阵浏览器）和 Vector Scope（向量示波器）。Matrix Viewer 将输入矩阵的行和列作为坐标轴，使用不同颜色表示矩阵元素的值，还可以根据需要自己建立一个颜色表。Vector Scope 显示输入的每一列（通道），按照指定帧的数目每次显示整个数据。Vector Scope 可以显示时域或频域信号。图 11.5 是基于帧的三个正弦信号（三个通道）分别用 Matrix Viewer 和 Vector Scope 显示的结果。此外还有内置 FFT 变换的 Spectrum Scope 用来直接显示时域信号的频谱。与 Simulink 中的信号 Sinks 库类似，DSP Blockset sinks 库还包含将信号写入一个工作区的块，唯一的差异是这里是基于帧的。

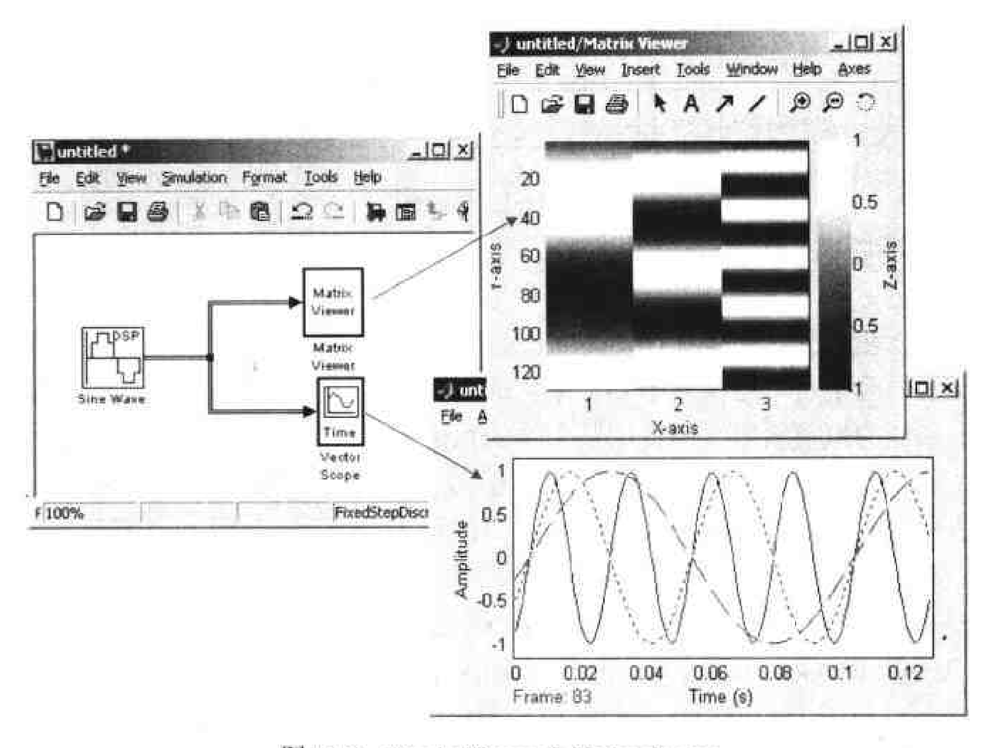


图 11.5 Matrix Viewer & Vector Scope

5. 使用基于帧的信号

当一个信号线表示基于帧的信号时，Simulink 用双线来绘制。基于帧的信号处理可以使用 Simulink 中对输入的每个元素进行处理的块，但是不能使用 Simulink 中对向量处理的模块（例如 Unit Delay 和 Mux）。实际上这些模块中许多模块在 DSP Blockset 中都有一个与之对应，专门用来做基于帧的信号处理的版本。例如，在 DSP Blockset 中等价于 Unit Delay 的模块是 Integer Delay 模块，与 Mux 等价的模块是 Matrix Concatenation 模块。图 11.6 所示的框图是对随机信号延迟 30 个步长后进行卷积处理。下面给出一个具有回响功能的声学例子，读者不妨一试。

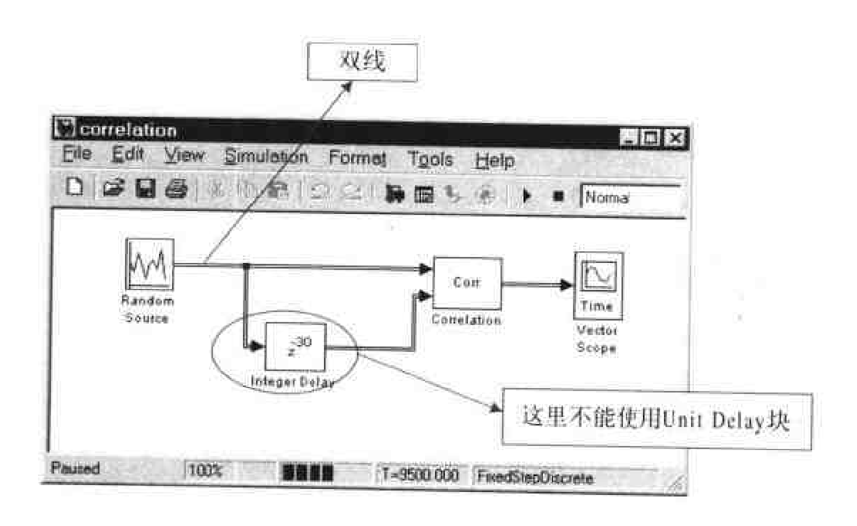


图 11.6　基于帧的信号处理

【例 11.1】　试建立一个圆形剧场的声学模型，假设有 70%的信号被反射回来。

解： 使用\DSP Sources\From Wave File 模块加载一个声音文件(*.wav)，采样频率为 8000 Hz。圆形剧场的回声效果导致 70%的信号在 2 s 之后反射回来，这里用一个增益为 0.7 的 Gain 模块表示。其中使用一个 Integer Delay 模块产生 2×8000 的采样延迟。最后使用一个\DSP Sinks\To Wave Device 模块听一下效果。回声系统模型如图 11.7 所示。注意，To Wave Device 模块只能在 PC 平台上使用。

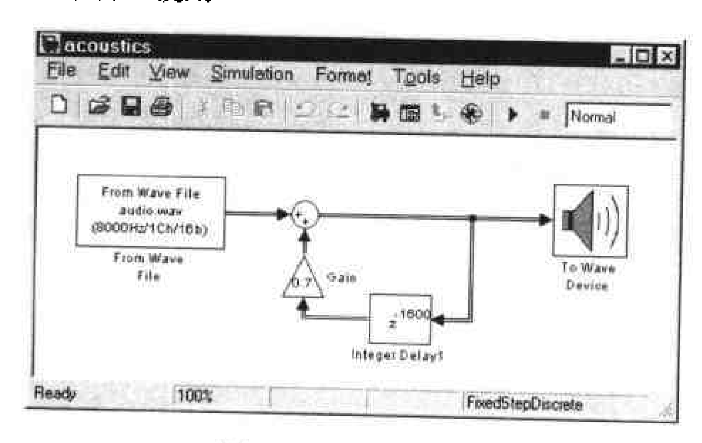


图 11.7　回声系统模型

11.1.2 设置 Simulink 进行 DSP 仿真

对于一般系统而言，Simulink 的默认设置认为信号是连续的，而且使用连续变步长求解器对系统进行求解。如果系统中包含连续信号和离散信号，应当使用此配置。但是对于纯离散的数字信号处理系统的设计、仿真与分析而言，需要对 Simulink 重新配置，使其能够适用于数字信号处理。

用户可以使用 M 文件 dspstartup 来配置 Simulink，使之适用于数字信号处理。设置的内容包括：使用固定步长求解器、在采样之间信号没有定义（避免两个不同采样率信号之间的操作）、结束时间设为无穷大、仿真时间和数据不保存到工作区以节省内存等等。此外，用户还可以根据需要修改 dspstartup.m 文件以加入定制的设置。图 11.8 为设置好的 DSP 仿真参数页面。

如果经常需要进行 DSP 仿真，用户可以在 startup.m 文件中加入 dspstartup 命令，MATLAB 在启动后自动运行 startup，这样就无需每次仿真都运行 dspstartup 命令了。

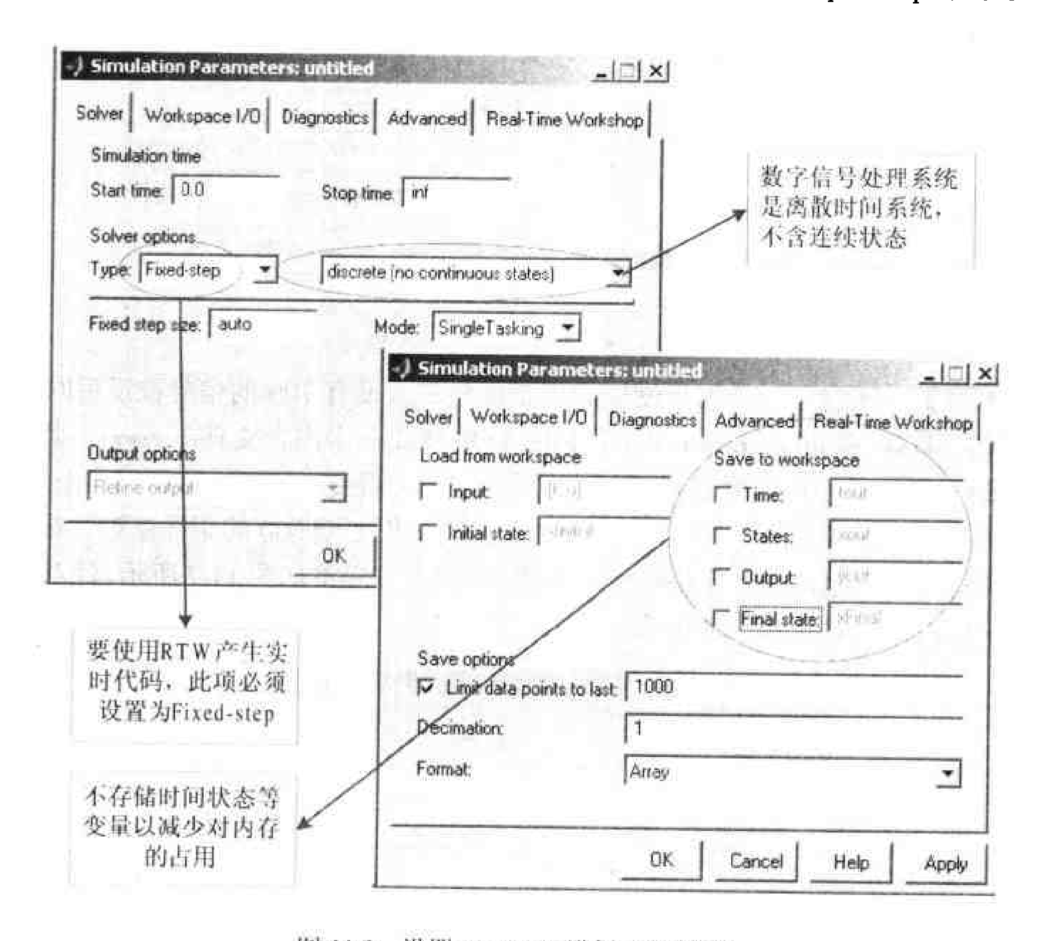


图 11.8 设置 Simulink 进行 DSP 仿真

11.2　DSP Blockset 模块库介绍

DSP Blockset 库提供了极为丰富的 DSP 模块资源，它们封装了几乎所有基本的数字信号处理操作和算法，其中的许多模块在信号处理工具箱中都有对应的函数。用户可以利用这些模块方便地完成自己的数字信号处理系统仿真和分析。图 11.9 列出了展开后的 DSP Blockset 模块库。这一节将分别介绍各个子库并给出一些简单的例子。

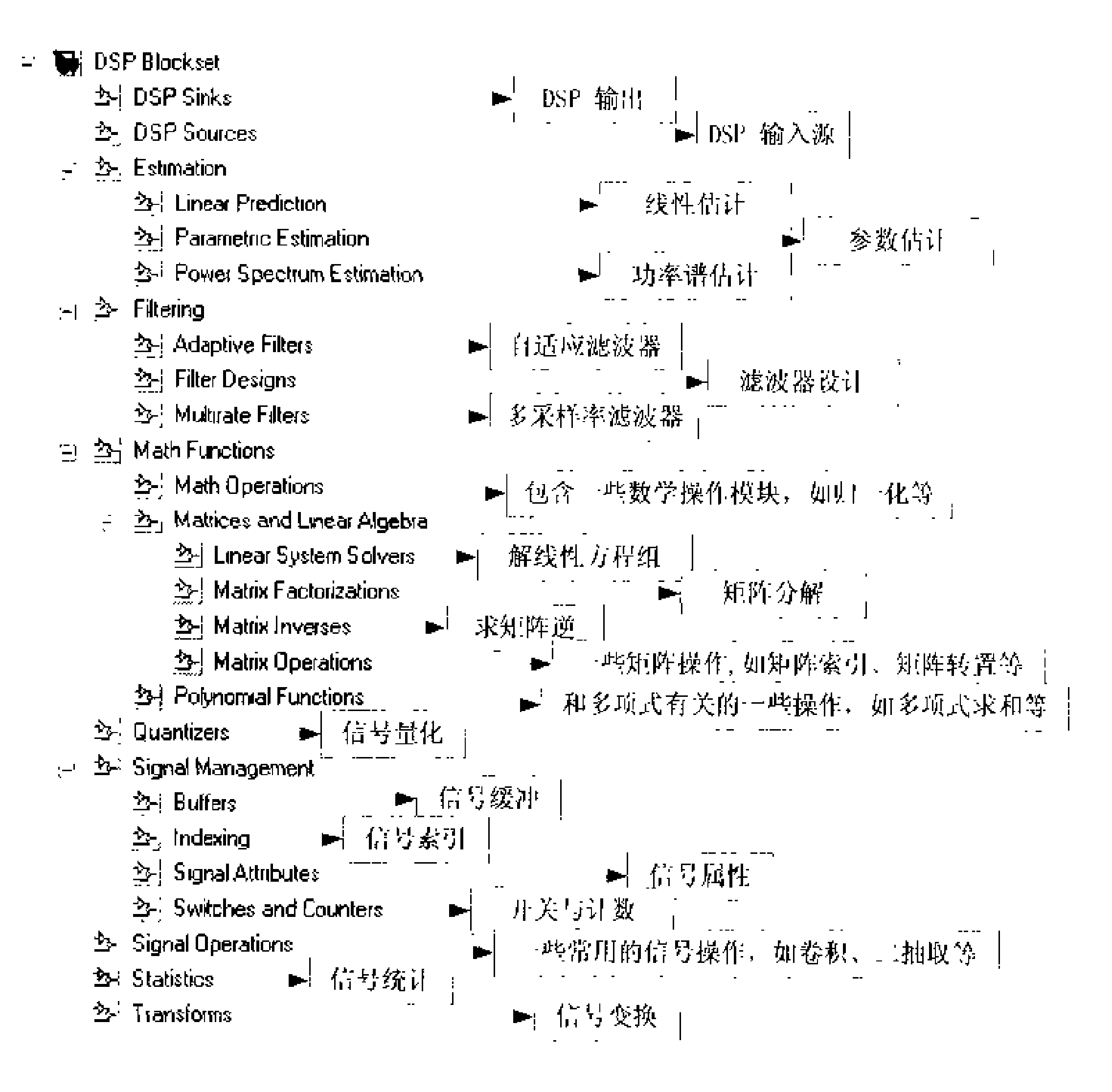


图 11.9　DSP Blockset 模块库

11.2.1　信号的操作和管理

一般的信号操作如加窗和补零可以通过 Signal Operations 模块库完成。这个库中还包含 Variable Interge Delay 模块，这里延迟的大小是通过第二个输入信号指定的。表 11.2 列出了各个 Signal Operations 库中各个模块及其功能描述。

表 11.2 Signal Operations 库

模块名称	模块功能描述
Convolution	计算两个输入的卷积
Downsample	抽取操作
Upsamole	插值操作
Interger Delay	延迟操作
Pad	补值操作
Zero Pad	补零操作
Repeat	重复操作，重复输入采样 N 次
Sample and Hold	当收到一个触发信号后，对输入信号采样并保持直到收到下一个触发信号
Unwrap	展开信号的相位
Variable Fractional Delay	按照一个变量的值延迟每个通道的信号，该变量可以是分数
Variable Integer Delay	按照一个变量的值延迟每个通道的信号，该变量是整数
Window Function	加窗操作，可以选择不同的窗类型

信号可以通过在 Signal Management 下的四个库进行一些管理操作，它包括缓冲、索引、信号属性、切换与计数四个部分，如表 11.3 所示。

表 11.3 Signal Management 库

子库名称	模块名称	模块功能描述
Buffers	Buffer	缓冲
	Unbuffer	解缓冲
	Delay line	重新缓冲信号，每次更新一个采样
	Queue	FIFO（先入先出）寄存器
	Stack	实现一个栈，或者先入后出寄存器
	Triggered Dealy line	带有使能端的 Delay line
Indexing	Flip	按行或者按列倒置矩阵或者向量
	Selector	从矩阵或者向量中选择元素
	Multiport Selector	从矩阵或者向量中选择多组元素
	Variable Selector	按照输入变量选择元素
	Submatrix	从一个矩阵中选择一个子矩阵
Signal Attributes	Check Signal Attributes	检查信号的属性是否符合设置
	Contiguous Copy	将非连续存储的信号转换为连续存储的信号
	Convert 1-D to 2-D	将一维信号转换为二维信号
	Convert 2-D to 1-D	将二维信号转换为一维信号
	Frame Status Conversion	设置输出帧的状态
	Inherit Complexity	根据参考信号改变输入信号的表示形式（复数或实数形式）
Switch and Counters	Counter	脉冲计数器
	Edger Detector	边缘检测器，当信号变为 0，或者从 0 变为其它值时，输出 1
	Event-Count Comparator	统计非零输入的个数，当大于设置的数目时，输出变为 1
	Multiphase Clock	产生相位依次移动的时钟信号阵列
	N-Sample Enable	经过 N 个采样后输出从 0 变为 1
	N-Sample Switch	经过 N 个采样后输出从下端口输入变为上端口输入

信号的速率转换可以通过 Signal Operations 中的 Upsample 和 Downsample 模块来进行。Upsample 通过在新的数据点上补零来实现，Downsample 通过间隔去除部分采样点来降低采样速率。如图 11.10 所示，用一个 probe 模块来探测信号的采样速率，原始采样信号为 1000 Hz，经过二抽取后变为 500 Hz，经过插值后变为 2000 Hz。

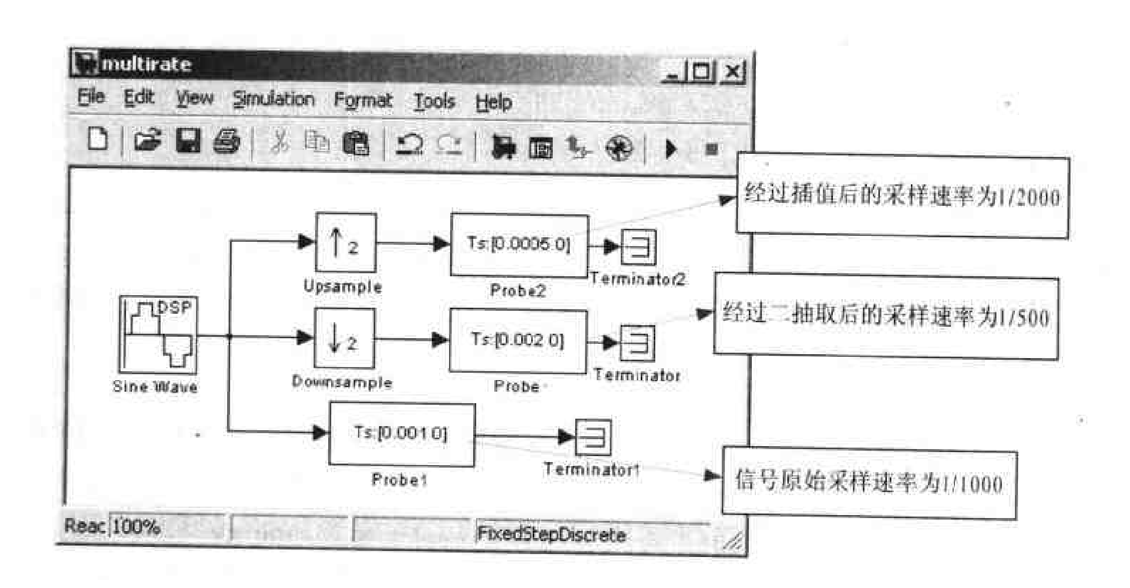


图 11.10　改变信号的速率

使用\DSP Blockset\Filtering\Multirate Filters 库中的 FIR Interpolation 模块进一步对输出进行滤波，生成插值数据。使用 Downsample 模块从输入信号中删除数据点。对于有些信号，可能由于镜像（aliasing）的作用导致信息的丢失。FIR Decimation 模块由于在下变频之前对信号滤波而避免了这个问题。所有的速率转换模块使用整数因子。如果需要做分数的转换，则应通过对信号先做上变频，再做下变频来完成。

有关信号速率的例子读者可以参考 Simulink 提供的例子：dspsrcnv.mdl。它分别提供基于采样的基于帧的处理的两个版本，说明了信号采样速率转换的多种实现方法。要打开该例子，应在 MATLAB 命令行下输入：

```
>>open dspsrcnv
```

或者

```
>> dspsrcnv
```

11.2.2　信号变换

使用基于帧的信号的一个好处就是能够进行各种变换处理，这意味着可以得到关于信号特性的更多信息。Simulink DSP Blockset 提供了时域至频域，频域至时域（逆变换）和时域至时域的转换模块。需要注意的是，这些变换模块只能用于基于帧的输入的场合。表 11.4 列出了 Transform 库中的模块及其功能描述。

表 11.4　Transform 库

模块名称	模块功能描述
DCT	计算每个通道的离散余弦变换
FFT	计算每个通道的快速傅立叶变换
IDCT	计算每个通道的离散余弦反变换
IFFT	计算每个通道的离散傅立叶反变换
Analytic Signal	计算每个通道的分析信号
Complex Cepstrum	计算每个通道的复倒谱
Real Cepstrum	计算每个通道的实倒谱
Magnitude FFT	计算每个通道输入的幅度谱或者幅平方谱

这些变换中最常用的和最重要的无疑是 FFT 和 IFFT，下面举一个用于计算频率响应的简单例子。

【例 11.2】　计算信号的频率响应。

解：有两种简单的方法可以用来计算信号的频率响应。一种是对输入信号做 FFT 变换，然后在一个 Vector Scope 中观察变换的结果，这时 Vector Scope 的输入域应设置为 Frequency。在 Vector Scope 模块前需要插入一个 Complex to Magnitude 模块，因为向量示波器期待的是一个实型输入。另外一个方法是将信号直接接到一个 FFT 示波器上，这个示波器先进行 FFT 变换和求平方，然后再显示。计算信号的频率响应的框图如图 11.11 所示。输入信号是两个频率分别为 50 Hz 和 100 Hz 的正弦信号：

$$u(t)=[\sin(100\pi t)\quad \sin(200\pi t)]$$

各个模块的参数设置如下：

(1) Sine wave 模块：Frequency(Hz)设置为[50 100]。

(2) Gain 模块：Gain 设置为 0.5。

(3) Complex to Magnitude 模块：Output 设置为 Magnitude。

(4) Vector Scope：Input domain 设置为 Frequency。

其它使用缺省设置。

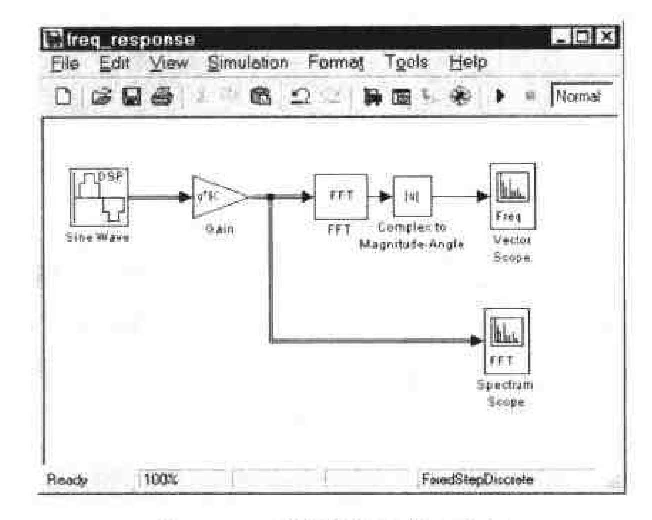


图 11.11　计算信号的频率响应

11.2.3　滤波器设计与频率分析

有关滤波器设计和分析的内容非常丰富，在此仅对\Simulink DSP Blockset\filtering 库中的模块进行简单的介绍，然后利用 Simulink 所提供的工具设计一个最简单的低通滤波器。

1. 设计一个滤波器

DSP Blockset 库中有丰富的滤波器设计模块，可以设计数字 FIR 和 IIR 滤波器、模拟 IIR 滤波器。对于滤波器设计，应给出滤波器阶数和截止频率。实际上滤波器设计模块是通过信号处理工具箱(Signal Processing Toolbox)进行滤波器设计的，然后返回一系列滤波器系数。设计好的滤波器会探测它的输入是基于采样的还是基于帧的，然后根据具体情况运行。对于基于帧的信号，滤波器沿输入的每一列进行操作。

Simulink 4.0 为用户提供了两个非常好用的模块用于模拟滤波器和数字滤波器的设计。Analog Filter Design 模块用于模拟滤波器的设计，只需在模块对话框中选择要设计的滤波器类型和方法及其它阶数等信息就可以了。Digital Filter Design 模块用于数字滤波器的设计，双击该模块可以看见如图 11.12 所示的图形化的设计界面，通过它用户可以方便地进行各种常用数字滤波器的设计和分析，设计完后可以直接作为滤波器的实现模块在仿真中使用。双击模块实际上打开的是 FDATool，在命令行下输入 FDATool 同样能够打开图 11.12 所示的界面，滤波器的实现则是用一个 S-函数实现的。下面我们使用该模块来设计并实现一个简单的低通数字滤波器。

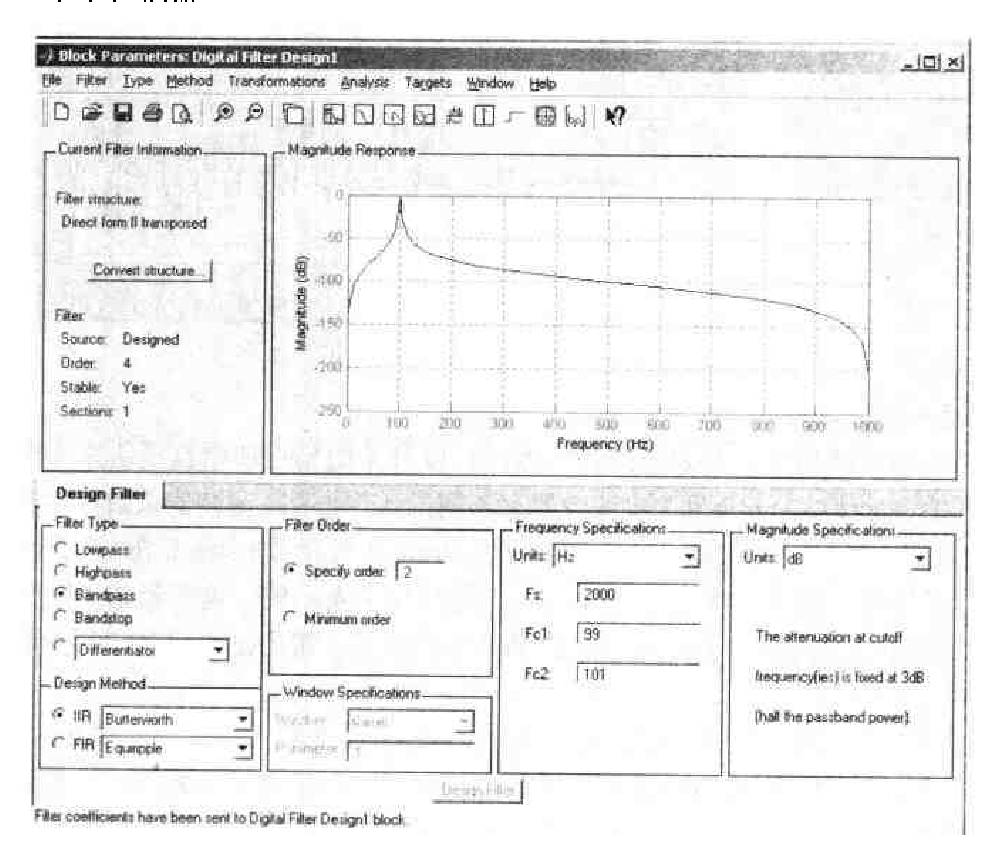


图 11.12　滤波器设计工具：FDATool

【例 11.3】　设计一个滤波器滤除正弦信号中的噪声，还原正弦信号。正弦信号为 $\sin 100t$，噪声信号是均值为 0、方差为 1 的高斯白噪声。信号采样频率为 2000 Hz。

解：(1) 首先用 Digital Filter Design 模块设计 Butterworth 带通滤波器。各个选项参数的设置如图 11.12 所示。然后单击 Design Filter 按钮，完成滤波器的设计。

(2) 按照图 11.13 选择和连接好其余各个模块。各个模块的设置如下：

① Random Input 模块：Source Type 设置为 Gaussian，Sample Time 设置为 1/2000，Samples per fame 设置为 256。

② Sine Wave 模块：Frequency 设置为 100，Sample Time 设置为 1/2000，Samples per fame 设置为 256。

③ Vector Scope 模块：为缺省设置。

运行仿真，产生的结果如图 11.13 所示。从图中可以看出，正弦信号被清楚地还原了出来。

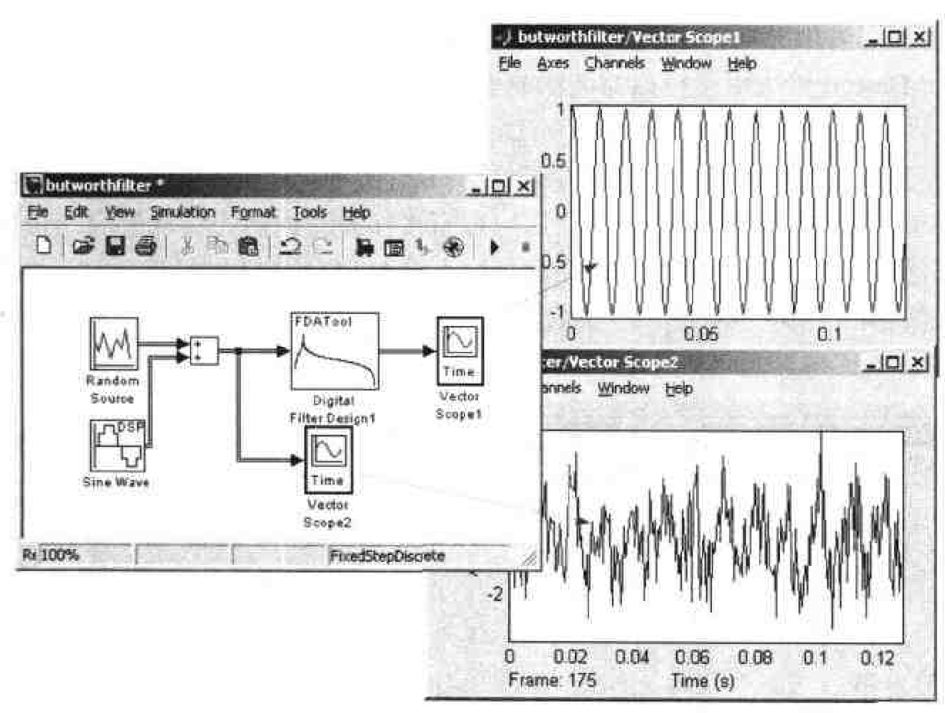


图 11.13　带通滤波器结果输出

2. 实现一个滤波器

通过一个给定的传递函数表达的滤波器，可以只采用延迟和增益模块来实现。这个实现不是唯一的，而且对于 FIR 和 IIR 滤波器有不同的实现方法。滤波器系数可以直接在滤波器实现块中输入，或者使用 Signal Processing Toolbox 中的函数 butter、fir1 等产生。注意，设计这些滤波器时，截止频率往往是通过归一化频率来表示的。这里最常用的一个模块就是 Direct-Form II Transpose Filter 块，它是滤波器的直接实现方法，可使用最少的单位延迟环节来实现。

当希望采用给定的构件实现滤波器时，请使用滤波器实现向导（Filter Realization Wizard）。使用滤波器实现向导只需指定实现方法和所使用的滤波器系数，该模块就能够自动生成所需要滤波器的实现。用户可以选择将实现的滤波器放在当前的模型中或在新的模型中并为生成的模块设置名字。用户还可以指定使用 Simulink 中基于采样的模块、DSP Blockset 中基于帧的模块，或者采用 Fixed Point Blockset，可以选择 1～128 位的位数。一旦做好选择，Filter Realization Wizard 就会尝试改变系数，优化所采用部件的数目。

Filtering 库中还有很多其它模块，在此不再一一叙述，请读者参考相关文献。

11.2.4　功率谱估计

Estimation 库提供了对输入信号进行功率谱分析的各种模块。信号的功率谱在仿真过程中只能通过估计得到，因为只能对有限长度的信号进行处理。功率谱估计主要有两类方法：非参数估计方法和参数估计方法。非参数估计方法直接使用信号进行功率谱估计；参数估计方法试图建立一个等效系统并估计出系数。\DSP Blockset\Estimation\Parametric Estimation 库给出了几种 AR 模型参数估计方法；\DSP Blockset\Estimation\Power Spectrum Estimation 库则给出了利用 Parametric Estimation 库中的参数估计模块进行功率谱估计的模块，以及两种利用非参数估计方法估计信号功率谱的模块。同滤波器的设计模块一样，许多模块所代表的方法在信号处理工具箱中都有对应的函数。如经典谱估计的周期图法在 DSP Blockset 中的实现模块是 Magnitude FFT 模块，在信号处理工具箱中对应的函数是 periodogram。表 11.5 列出了各个模块的名称及其功能描述。

表 11.5　Statistics 库

模块名称	模块功能描述
Autocorrelation LPC	自相关线性预测
Yule-Walker AR Estimator	利用 Yule-Walker 方法估计 AR 模型参数，它使用 Levinson-Durbin 递推算法解 Yule-Walker 方程
Burg AR Estimator	利用 Burg 方法估计 AR 模型参数
Covariance AR Estimator	利用协方差方法估计 AR 模型参数
Modified Covariance AR Estimator	利用改进的协方差法估计 AR 模型参数
Burg Method	利用 Burg 方法估计信号功率谱，它用到了 Burg AR Estimator 块
Covariance Method	利用协方差方法估计 AR 模型参数，它用到了 Covariance AR Estimator 块
Modified Covariance Method	利用改进的协方差方法估计 AR 模型参数，它用到了 Modified Covariance AR Estimator 块
Yule-Walker Method	利用 Yule-Walker 方法估计信号功率谱，它用到了 Yule-Walker AR Estimator 块
Magnitude FFT	利用直接法（周期图法）估计信号功率谱
Short-Time FFT	利用短时傅立叶变换估计信号功率谱

【例 11.4】　让一段零均值功率为 σ^2 的白噪声通过一 AR 模型：

$$H(z)=\frac{1}{1-0.1z^{-1}+0.09z^{-2}+0.648z^{-3}}$$

得到输出 $y(n)$，再加上一个频率为 350 Hz 的实正弦信号 $x(n)$，然后利用 Power Spectrum Estimation 库中的各种方法估计上述信号的功率谱。

解：(1) 首先绘制 $x(n)+y(n)$ 的实际功率谱。设白噪声信号均值为 0，方差为 0.1。通过 $H(z)$ 后功率谱为 $P_w(\omega)=\sigma^2|H(\omega)|^2$，从而可得 $y(n)$的实际功率谱。图中椭圆框内是计算 $y(n)$ 功率谱的模块。已知正弦信号功率谱为在 350 Hz 处的线谱,从而可得上述信号的实际功率谱如图 11.14 中未标记的实线所示(图 11.14 中正弦信号的实际功率谱为后来手工绘制上去的)。

(2) 将 $x(n)+y(n)$ 信号缓冲成为帧后，再送往各个谱估计模块，三个估计结果和 $y(n)$ 的真实谱送往矩阵拼接模块，以使我们可以在一个示波器内观察对比不同的输出结果。

各个模块及其参数设置如下：

① White Noise 模块：Source Type 设置为 Gaussian，Variance 设置为 0.1。

② Sine Wave 模块：Frequency 设置为 350。

③ Direct-Form II Transpose Filter 模块：用于表示 $H(z)$ 。Numerator 设置为 1，Denominator 设置为[1 -0.1 0.09 0.648] 。

④ Overlap Buffer 模块：Output buffer size 设置为 128，Buffer overlap 设置为 64。

⑤ Burg Method 模块：Estimation order 设置为 6，FFT length 设置为 128 。

⑥ Modified Covariance Method 模块：Estimation order 设置为 6，FFT length 设置为 128。

⑦ Yule-Walker Method 模块：Estimation order 设置为 6，FFT length 设置为 128。

⑧ Matrix Concatenation 模块：Numbet of inputs 设置为 4，Concatination method 设置为 Horizonal 。

以下几个模块可计算白噪声经过 AR 模型的功率谱。

① Constant 模块（设置 AR 模型系数）：Constan value 设置为[1 -0.1 0.09 0.648]。

② Magnitude FFT 模块：FFT length 设置为 128。

③ Math Function 模块：Function 设置为 Reciprocal。

④ Gain 模块：Gain 设置为 0.1。表示白噪声的方差，即白噪声功率谱幅值。

输出结果如图 11.14 所示，读者不妨改变几个功率谱估计模块的阶次，观察输出结果会有什么变化。

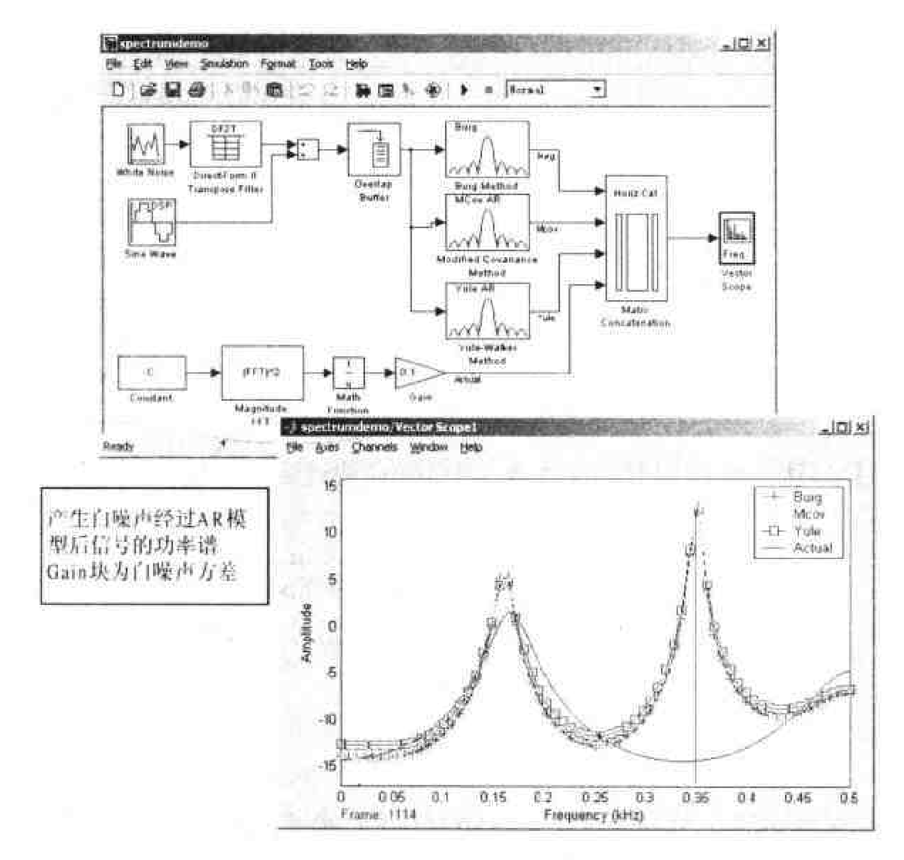


图 11.14　功率谱估计

11.2.5　统计

Statistics 库提供了一些常用统计操作模块，如求最小值、最大值、平均值、方差等。大部分模块都支持两种模式：基本模式和运行期模式。基本模式的统计对象是当前仿真周期内的输入，与之前的输入无关，这个输入可以是基于采样的也可以是基于帧的，当然对于每个通道只有一个值的输入，统计操作就没有任何意义了。运行期模式不仅统计当前的输入还涉及到过去的输入。用户可以在 RST 端口上设置一个 reset 信号来控制每次统计的起始位置。需要注意的是，Minimum 和 Maximum 两个模块，由于其运行期模式不可能进行索引操作，因此以下拉菜单的形式给出了 Value and Index, Value, Index 和 Running 四个选项，其中前三者属于基本模式。表 11.6 列出了 Statistic 库中的各个模块的名称及其功能描述。

表 11.6　Statistics 库

模块名称	模块功能描述
Maximum	求最大值，有基本模式和运行期模式两种模式
Minimum	求最小值，有基本模式和运行期模式两种模式
Median	求中间值，有基本模式和运行期模式两种模式
Sort	排序，使用快速排序算法，只有基本模式
Mean	求均值，有基本模式和运行期模式两种模式
Variance	求方差，有基本模式和运行期模式两种模式
Standard Deviation	求标准方差，有基本模式和运行期模式两种模式
RMS	求均方根，有基本模式和运行期模式两种模式
Correlation	求相关，只有基本模式
Autocorrelation	求自相关，只有基本模式
Histogram	统计直方图，有基本模式和运行期模式两种模式
Detrend	去除信号中的趋势项

【例 11.5】 统计一个高斯分布的随机信号的方差、均值并绘制其直方图。

解：Simulink 框图如图 11.15 所示。所需的各个模块及其参数设置如下：

(1) Random Source 模块：Source type 设置为 Gaussian，Samples per frame 设置为 100。

(2) Histgram 模块：Minimum value of input 设置为-10，Maximum value of input 设置为 10，bin 设置为 21。

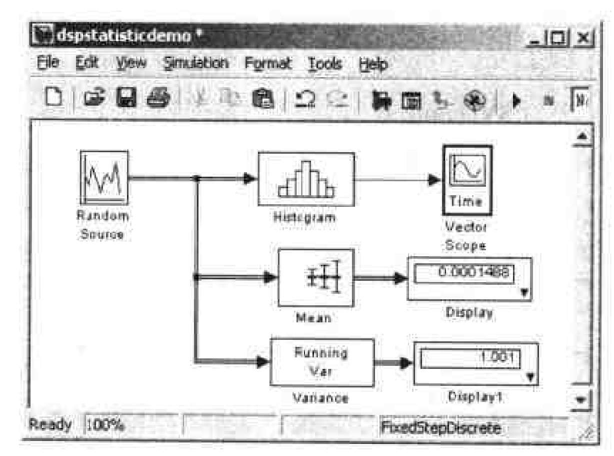


图 11.15　信号统计

在 Mean 模块、Variance 模块和 Histogram 模块中选中或者去除 running 复选框，对比观察不同的实验结果，可以看到当使用 running 模式时显示的实验结果更稳定，更接近于真值，因为它统计的是从开始到当前的所有输入。

11.2.6 矩阵操作与线性方程求解

数字信号处理中经常会遇到多通道信号，DSP Blockset 包含了一个 \Math Functions\Matrices and LinearAlegbra 库实现对多通道信号（矩阵信号）的处理。常用的操作包括对行或列求和或求积、矩阵乘积和对角线提取等，这些操作包含在 Matrix Operations 库中， 前面介绍的 Matrix Concatenation 模块也在这个库中。另外，Matrix Factorizations 库实现了许多 MATLAB 矩阵分解算法供仿真使用；Matrix Inverses 库封装了几种求矩阵逆的算法；Linear System Solvers 库包含求解 AX=B 类型的线性方程组的若干方法。表 11.7～11.9 列出了包含的模块及其功能描述。

表 11.7 Matrix Factorizations 库

模 块 名 称	算 法 描 述
LU Factorization	LU 矩阵分解
Cholesky Factorization	Cholesky 矩阵分解
LDL Factorization	LDL 矩阵分解
QR Factorization	QR 矩阵分解
Singular Value Decomposition	SVD 矩阵分解

表 11.8 Matrix Inverses 库

模 块 名 称	算 法 描 述
LU Inverse	利用 LU 分解算法求矩阵的逆
Cholesky Inverse	利用 Cholesky 分解算法求矩阵的逆
LDL Inverse	利用 LDL 分解算法求矩阵的逆
Pseudoinverse	利用 SVD 分解算法求矩阵的伪逆

表 11.9 Linear System Solvers 库

模 块 名 称	算 法 描 述
LU Solver	利用 LU 分解算法解线性方程组，A 阵必须为方阵，B 必须与 A 有相同的行数
SVD Solver	利用 SVD 分解算法解线性方程组，如果 A 不是方阵，则所得解为最小二乘解
QR Solver	利用 QR 分解算法解线性方程组，如果 A 不是方阵，则所得解为最小二乘解
Levinson-Durbin	利用 Levinson 递推算法求解 Toeplitz 型方程组
LDL Solver	利用 LDL 分解算法解线性方程组，A 为对称正定的方阵，B 与 A 有相同的行数
Cholesky Solver	用 Cholesky 分解法求（即平方根法）解线性方程组，A 为对称正定的方阵，B 与 A 有相同的行数
Forward Substitution	前向替代法解线性方程组，其中 A 是下三角方阵
Backward Substitution	后向替代法解线性方程组，其中 A 是上三角方阵

【例 11.6】　利用 LU 分解求解线性方程组 AX=B 的解。其中 A=[1 -2 3; 4 0 6 ; 2 1 5], B=[1 –0.225　–1; 1 2 3]。

解：(1) 利用 DSP Constant 模块输入 A 阵和 B 阵 。

(2) 在 Linear System Solvers 库中选择 LU Solver 模块，它使用 LU 分解算法求解方程组的解。

(3) 加入 display 模块用于显示方程组的解。

按照图 11.16 所示，连好连线，并仿真得到上述方程的解，并在 display 模块中显示出来。注意由于 B 是 3×2 矩阵，所以求得两组解（两个列向量）。我们来验证一下解的正确性，将解代入 AX，这里用一个 Matrix Multiply 来实现矩阵的乘法，求得的结果显示在 display1 中，正是 B 阵。注意这些运算只需一个仿真步长就可以完成，所以仿真时间在这里就没有意义了，因为图 10.16 本身所表示的框图是一个静态系统，在数学上是一组代数方程。

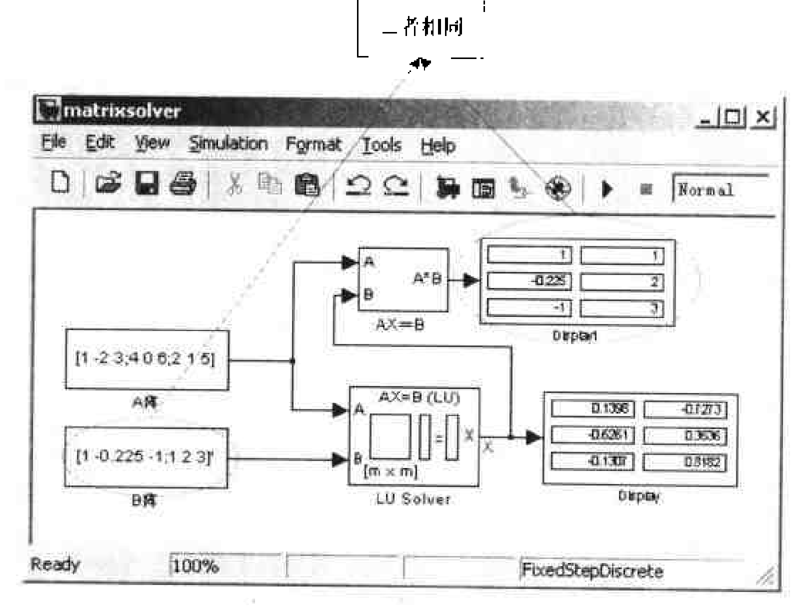


图 11.16　利用 LU 分解求解线性方程组



习　题

1. 生成一个基于帧（64 个采样）的信号：u(t)=sin(2*pi*20*t)+0.25sin(2*pi*400*t)，采样频率 F_s=1000。将信号输入下面差分方程定义的系统中：

$$y(n)=1/2*(u(n)+u(n-1))$$

然后使用向量示波器（Vector Scope）观察原始信号和结果信号。若可能的话，请使用尽量少的模块实现上述处理。

2. 设有信号 $x_i(n)$, $n=0,1,2\cdots,N-1$; $i=1,2,\cdots,m$，使用这 m 个信号来近似信号 $y(n)$，即 $\hat{y}(n)=a_1x_1(n)+a_2x_2(n)+\cdots+a_mx_m(n)$，试用 Simulink 求向量 $\boldsymbol{a}=[a_1,a_2,\cdots,a_m]$，使得 $\hat{y}(n)$ 对 $y(n)$ 近似的平方误差为最小，即 $E=\sum_{i=1}^{m}\left|y(n)-\hat{y}(n)\right|^2$ 为最小。

提示：要使 E 最小，只需 $\frac{\partial E}{\partial \boldsymbol{a}}=0$，由此可求得 $\boldsymbol{a}$ 的估计 $\hat{\boldsymbol{a}}=(\boldsymbol{X}^H\boldsymbol{X})^{-1}\boldsymbol{X}^H\boldsymbol{y}$。然后使用 Matrices and Linear Alegbra 库中的矩阵相乘、矩阵求逆模块实现上式。

3. 在 PC 机上用任意播放器播放音乐，然后使用 Simulink DSP Blockset 观察所播放音乐信号的频谱。

提示：使用 From Wave Device 获得 PC 音频信号，然后估计其功率谱。

附录 内容索引

参 考 文 献

【1】 MathWorks. SIMULINK，Using Simulink. 2001
【2】 MathWorks. SIMULINK，Writing S-Functions. 2001
【3】 MathWorks. DSP Blockset User's Guide. 2001
【4】 MathWorks. MATLAB，The Language of Technical Computing. 2001
【5】 MathWorks. MATLAB Compiler. 2001
【6】 王正中，屠仁寿著. 现代计算机仿真技术及其应用. 北京：国防工业出版社，1991
【7】 康凤举主编. 现代仿真技术与应用. 北京：国防工业出版社，2001
【8】 薛定宇著. 科学运算语言 MATLAB5.3 程序设计与应用. 北京：清华大学出版社，2000
【9】 郑大钟编. 线性系统理论. 北京：清华大学出版社，1993
【10】 阙志宏编. 线性系统理论. 西安：西北工业大学出版社，1995
【11】 胡广书编著. 数字信号处理——理论、算法与实现. 北京：清华大学出版社，1997
【12】 薛定宇著. 反馈控制系统设计与分析. 北京：清华大学出版社，2000
【13】 薛定宇著. 控制系统设计与分析. 北京：清华大学出版社，2000
【14】 李人厚，张平安等译校. 精通 MATLAB——综合辅导与指南. 西安：西安交通大学出版社，1998
【15】 楼顺天. 基于 MATLAB 的系统分析与设计——信号处理. 西安：西安电子科技大学出版社，1998